Titre: Title:	Réalisation mixte matérielle/logicielle d'un algorithme de routage pour réseaux locaux
Auteur: Author:	Mohamed Tahar Haddad
Date:	1996
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Haddad, M. T. (1996). Réalisation mixte matérielle/logicielle d'un algorithme de routage pour réseaux locaux [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8998/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8998/
Directeurs de recherche: Advisors:	
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

RÉALISATION MIXTE MATÉRIELLE/LOGICIELLE D'UN ALGORITHME DE ROUTAGE POUR RÉSEAUX LOCAUX

MOHAMED TAHAR HADDAD

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

DÉCEMBRE 1996

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE

Ce mémoire intitulé:

RÉALISATION MIXTE MATÉRIELLE/LOGICIELLE D'UN ALGORITHME DE ROUTAGE POUR RÉSEAUX LOCAUX

présenté par : HADDAD Mohamed Tahar

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. ABOULHAMID El-Mustapha, Ph.D., président

Mme. KAMINSKA Bozena, Ph.D., membre et directeur de recherche

M. BOIS Guy, Ph.D., membre et codirecteur de recherche

M. BLAQUIERE Yves, Ph.D., membre

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, le professeur Bozena Kaminska pour m'avoir accepté au sein de sa grande équipe, pour l'intérêt qu'elle a montré à ce sujet de recherche et sa disponibilité tout au long de ce travail.

Je tiens aussi à remercier mon codirecteur de recherche, le professeur Guy Bois pour son soutien financier, ses conseils et les longues heures de discussions qui ont guidé ce travail.

Je tiens à exprimer mes remerciements chaleureux à mes collègues Chokri, Naim, Ghyslaine et aux nombreuses personnes qui, de près ou de loin, m'ont permis de mener à bien ce projet d'envergure.

Mes remerciements les plus vifs vont à ma femme Saida et mes enfants Zied, Soumaya, Dhouha et Ali qui m'ont soutenu avec leurs encouragements. Je sais qu'étant étudiant à temps plein, on ne peut être que mari et père à temps partiel. Sans votre amour et vos sacrifices, ce travail ne serait jamais avancé.

SOMMAIRE

Sous l'impulsion de progrès rapide et extraordinaire de la technique de transmission et des réseaux de communications, la tendance est d'adapter les protocoles de communication aux exigences sans cesse grandissantes en performances (débits de communication) ainsi qu'aux nouveaux supports.

Les protocoles de communications actuels sont formés de couches majoritairement logicielles. La matérialisation de certaines de leurs fonctionnalités constitue une décision qui affecte grandement le débit. Le résultat final vise des implémentations mixtes matérielles/logicielles qui favorisent un grand débit de communication tout en garantissant un minimum de fiabilité et de flexibilité. Le partitionnement matériel/logiciel choisi permet d'envisager l'implémentation d'un certain nombre de protocoles de hautes performances (Frame Relay, X25, etc.).

Ce travail vise donc à apporter une contribution à l'étude des protocoles de communication. Nous y trouvons la modélisation et l'implémentation du protocole Frame Relay. Nous proposons dans ce mémoire une méthodologie de modélisation de protocoles de communication utilisant: VHDL et C pour la spécification de protocole; MentorGraphics et Synopsis pour sa validation et sa synthèse de haut niveau. L'un des apports originaux de ce mémoire est sans doute la description mixte des principales fonctionnalités de Frame Relay, ce qui a permis d'avoir une accélération de débit assez importante pour certaines topologies de réseaux. Une comparaison des performances relatives à différents partitionnements vient appuyer cette hypothèse.

ABSTRACT

Due to the ongoing progress in transmission technology and networking, all communication protocols try to fit the increasing performance (communication throughput) requirements. This effort is often handicaped by the lack in high level description languages as well as the increasing complexity of modern protocols.

Present communication protocols are designed in a layered way. Most of the layers are software based, and converting their functions into hardware may help in increasing performances. The final result targets heterogeneous (Hardware/Software) implementations with high throughput while guaranteeing reliability and flexibility. The partitioning considered in our project produced an architecture that can support several high performance protocols (Frame Relay, X25, etc..).

In order to achieve a high communication throughput, this thesis presents a heterogenous implementation of communication protocols in which a protocol modeling and implementation are included. VHDL and C are used for protocol specification. While MentorGraphics and Synopsis are used for design validation and the high level synthesis. Finally, one of the originalities of this work is the heterogeneous description of Frame Relay main functions, which allowed an important throughput increase for most network topologies. A comparison studies of communication performance regarding of functionality partitioning has been performed. This study shows the effeciency of the proposed architecture in providing high throughput communication system.

TABLE DE MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

Les domaines de la spécification, la conception, et la synthèse de systèmes mixtes matériel/logiciel (M/L) sont en pleine expansion et de plus en plus populaires. De nos jours, la complexité de la conception des systèmes dédiés de haute performance, surtout celle comprenant des systèmes hétérogènes M/L, nécessite de nouvelles approches qui supportent à la fois des comportements complexes et des communications de haut niveau.

Le problème principal dans les approches courantes d'implémentations mixtes M/L réside dans l'intégration des deux solutions obtenues. L'usage d'un modèle fixe de communication M/L restreint l'applicabilité de cette approche. La non-disponibilité de modèles de communication de haut niveau mène à entamer la conception avec une description trop détaillée. La solution est, incontestablement, de trouver un modèle d'abstraction de protocoles de communications complexes durant le processus de conception. Nous devons être en mesure de supporter un modèle de communication à différents niveaux d'abstractions dans le but de retarder autant que possible la sélection du protocole de communication qui sera utilisé. Bien entendu, le but ultime est de permettre la réutilisation de modèles de communication existants pour permettre la synthèse M/L et l'implémentation (*mapping*) sur des plates-formes d'architectures existantes.

1.1 Objectifs généraux

Ce mémoire est une suite aux travaux qui ont été réalisés par Jerraya (1994), Gajski (1994), Antoniazzi (1990), Buchenrieder, Sedlmeier et C.Veith (1993) et Assi (1994). Toutes ces contributions visaient à trouver un modèle qui simplifierait la conception ou le co-design de systèmes de communications. Leurs méthodologies varient entre la solution purement matérielle pour Assi et la solution purement logicielle au départ pour Jerrraya. Les autres viennent se placer à un compromis des deux approches qui convergera finalement vers une solution optimale où le meilleur rapport coût/performance est recherché.

1.2 Objectifs spécifiques

Les réseaux locaux (LANs) opérant à des débits de 100 Mbits/s sont déjà disponibles commercialement, et certains travaux de recherches proposent des réseaux à débit supérieur à 1 Gbit/s. De nouvelles contraintes viennent accroître ce besoin sans cesse grandissant d'augmenter les performances des réseaux actuels, surtout avec l'avènement des applications multimédia (fig.1-1). En effet, les systèmes de communication actuels exigent des protocoles de plus en plus performants capables de fournir de hauts débits tout en garantissant la fiabilité de la transmission. Ces protocoles sont répartis sur plusieurs couches dont uniquement la première est matérielle. Cette configuration en couches majoritairement logicielles provoque un alourdissement des transferts de communications, sachant que le logiciel est généralement plus lent que le matériel. Toutefois, une matérialisation totale d'un système de communication est impossible à l'heure actuelle en raison de la rigidité des langages de description matérielle comparés aux langages de programmation logiciel.

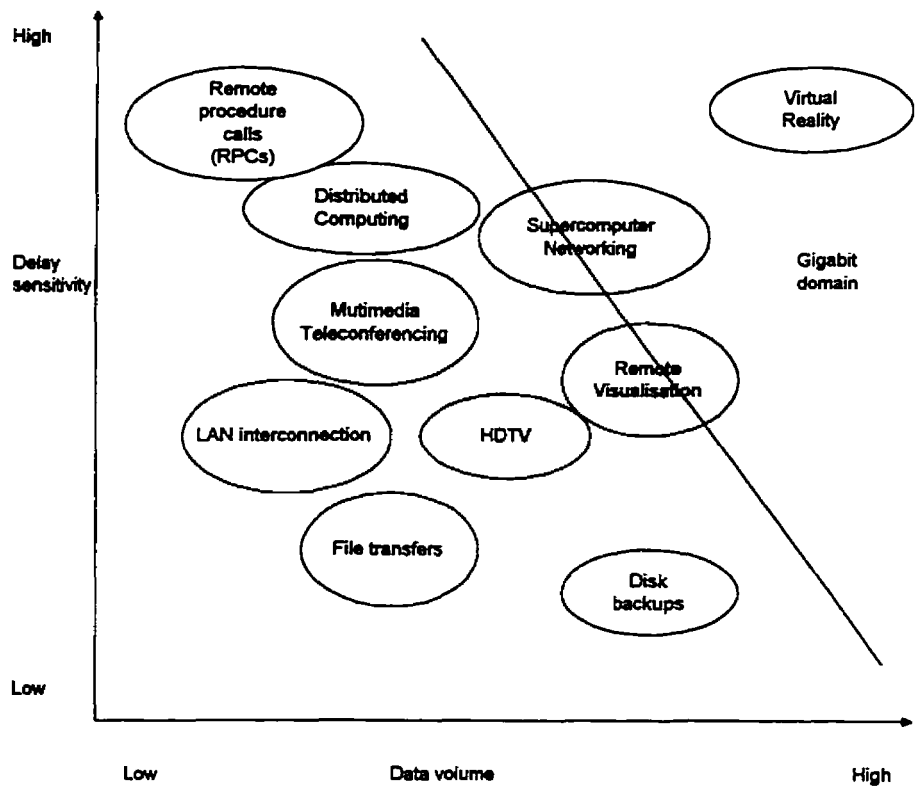


Figure 1-1: Exigences des applications à hautes performances.

Dans ce mémoire, nous nous proposons de développer et d'implémenter, d'une manière mixte M/L, un algorithme de routage pour un protocole de haute performance. Nous avons choisi le protocole Frame Relay étant donné les multiples avantages qu'il procure. Frame Relay est très bien adapté aux supports de communication modernes (fibre optique). Ce protocole opère sur 3 couches de communication dont seule la première est matérielle. Pour des raisons de coût et de maintenance, les deux autres couches ont toujours été implantées en logiciel, ce qui engendre une diminution du débit de transfert de données. Notre approche vise à optimiser le coût de l'implémentation matérielle pour certaines fonctions relatives à ces deux couches (sections de codes critiques), et à implémenter en logiciel la partie restante dans un ou plusieurs processeurs.

Voici en résumé les objectifs qui caractérisent l'approche de partitionnement présentée dans ce mémoire:

- Exploration étendue de l'espace des solutions.
- Estimation des performances pour la solution retenue.
- Adaptation facile à d'autres protocoles de haute performance.

1.3 Méthodologie

Notre approche de partitionnement suivra les étapes suivantes (figure 1.2)

- Un partitionnement initial est effectué. Deux grands blocs sont nettement séparés: un bloc matériel et un autre logiciel. Le premier sera exprimé en VHDL alors que le second le sera en C. Les deux blocs viendront remplir les différents constituants du système et seront générés par l'outil SDS de la compagnie MentorGraphics.

- Ce partitionnement sera raffiné, relativement à des fonctions de coûts bien définies, au fur et à mesure que nous avançons dans le projet.

- La validation de l'ensemble s'effectuera en simulant chaque bloc séparément, puis la simulation de l'ensemble suivra.

- Les performances (débit) seront comparées à celles d'un réseau local conventionnel opérant sous Frame Relay.

- La synthèse des différents blocs en FPGA et en technologie CMOS 1.2 µm sera la partie concluant ce travail.

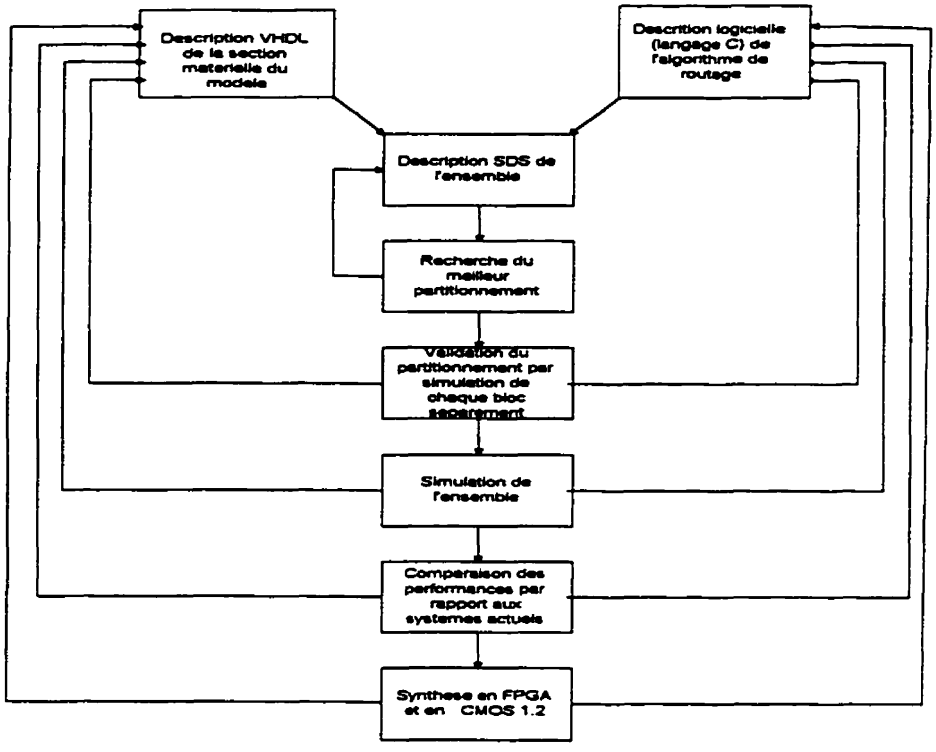


Figure 1-2: Méthodologie suivie.

1.4 Résultats anticipés

Notre travail vise, entre autres, à créer un modèle de réseau capable de supporter des protocoles de communication divers tout en matérialisant des tâches qui se passent actuellement en logiciel. En conséquence, le débit atteint doit être supérieur au débit présent.

Cette approche consistera donc à mesurer les débits pour les solutions de partitionnement suivantes:

1. Seule la première couche est matérielle. Il s'agit ici du modèle déjà existant sur le marché et pour lequel le logiciel est très sollicité. Ces résultats constitueront la borne minimale pour le débit désiré.

2. Tout est complètement matériel. Ce cas est évidemment utopique, mais ces résultats constitueront une borne maximale pour le débit désiré.

3. On cherche le maximum de matérialisation de tâches. C'est l'objectif même de ce travail. Les résultats obtenus seront un compromis entre les deux des points 1 et 2.

1.5 Plan du mémoire

Le chapitre 2 du présent mémoire fournit un aperçu sur les réseaux de communications, le protocole Frame Relay et la notion de routage dans les réseaux locaux de communication. Le chapitre 3 présente quelques notions de base relatives à la synthèse de haut niveau et au co-design. La chapitre 4 décrit l'approche de co-design considérée ainsi que les étapes suivies au cours de l'implémentation des différents blocs formant notre système. Le chapitre 5 résume les résultats pratiques relatifs à l'implémentation et à la simulation du système. Une comparaison des performances de différentes solutions suivra. Finalement, le chapitre 6 conclue ce mémoire en commentant les différents résultats obtenus.

CHAPITRE 2

SYSTÈMES DE COMMUNICATION

L'étude des protocoles de communication et de leurs complexités inhérentes sont à l'ordre du jour. Leurs réalisations, souvent longues, se heurtent à de nombreux problèmes de spécification, de mise en oeuvre et de test.

Dans ce chapitre, nous décrivons les concepts de base relatifs aux protocoles de communications, aux réseaux locaux de communications ainsi qu'à la notion de routage au sein de ces réseaux.

2.1 Concepts de base

Les réseaux de communication comportent une telle complexité qu'il est essentiel de structurer leurs fonctions. C'est ce que l'architecture OSI (Open System Interconnexion) essaie de faire. Cette architecture a été développée en premier lieu pour l'ISO (International Standard Organization), un organisme international de standardisation, et puis reprise par le CCITT.

L'architecture OSI est organisée en série de couches ou niveaux, chacune étant construite sur la précédente. Chacune de ces couches couvre des fonctions particulières. Il y a sept couches dans le modèle OSI dont les noms sont présentés à la figure (2-1).

Dans un réseau de communication, chaque noeud qui le compose comprend les sept couches du modèle OSI.

Lorsque deux noeuds communiquent entre eux, la couche *n* d'un noeud communique avec la couche *n* de l'autre noeud. Les règles et conventions utilisées dans cette communication sont connues sous le nom de protocole de la couche *n* (Fig. 2-2). Les entités comprenant les couches correspondantes sur différents noeuds sont appelées processus pairs. Ce sont elles qui communiquent à l'aide du protocole.

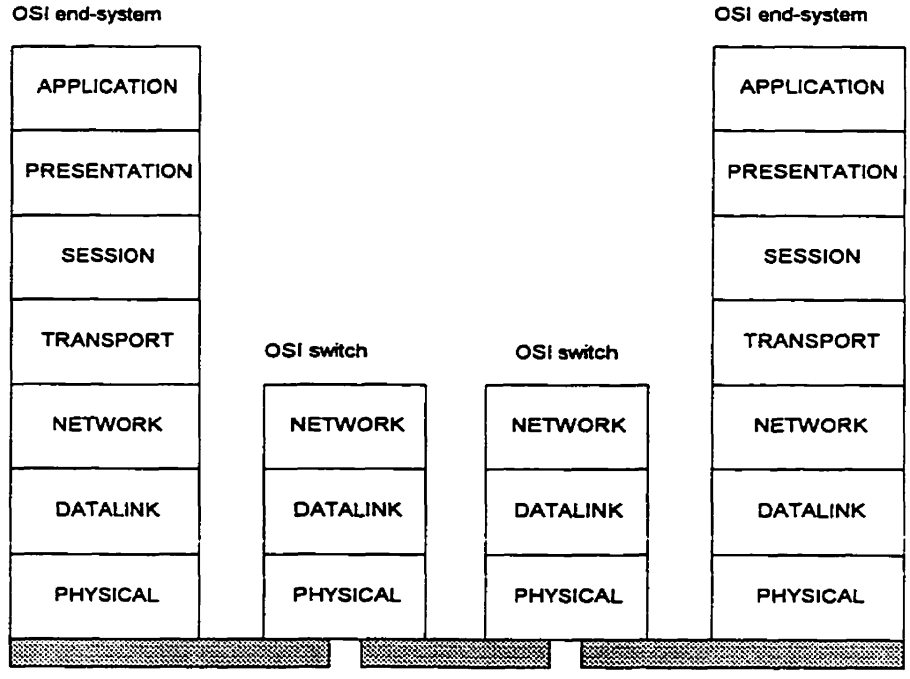


Figure 2-1: Les couches de communication du système OSI.

En réalité, aucune donnée n'est transférée directement de la couche *n* d'un noeud à la couche *n* d'un autre noeud. Chaque couche passe les données et le contrôle à la couche immédiatement inférieure, jusqu'à la plus basse. En dessous de la couche 1 se trouve le support physique qui véhicule réellement la communication.

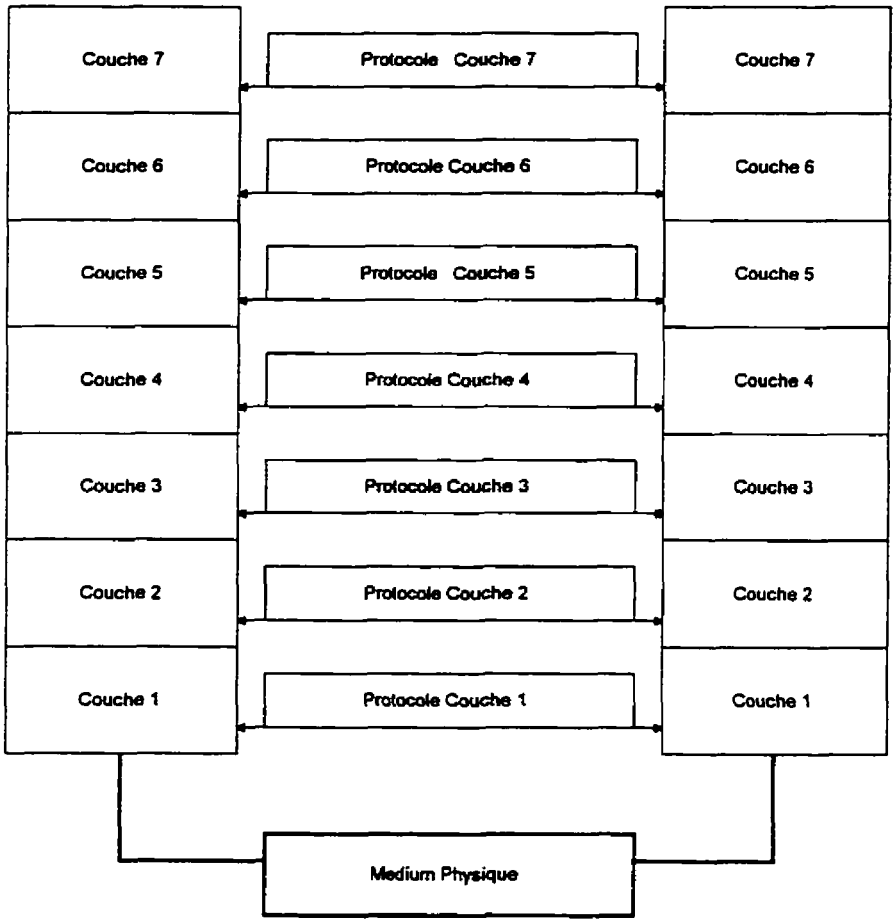


Figure 2-2: Protocoles des différentes couches.

Au noeud récepteur, chaque couche passe les données et le contrôle à la couche immédiatement supérieure, jusqu'à la plus haute. Le protocole entre deux couches *n* se fait par communication virtuelle. Les couches adjacentes d'un même noeud sont reliées par une interface. Les fonctions d'interfaces entre couches s'appellent 'services' et les commandes utilisées dans ces interfaces sont appelées 'primitives'.

L'interface définit les opérations élémentaires et les services que la couche inférieure offre à la supérieure. L'ensemble des couches et protocoles s'appelle l'architecture du réseau.

Le modèle OSI ne concerne pas l'architecture interne des systèmes, mais leur comportement externe permettant la communication. Les principes qui ont conduit à l'élaboration de sept couches sont les suivants:

- Une couche doit être créée lorsqu'un nouveau niveau d'abstraction est nécessaire.
- Chaque couche exerce une fonction bien définie.
- Les fonctions de chaque couche doivent être choisies en pensant à la définition de protocoles normalisés internationaux.
- Le choix de frontières entre couches doit être suffisamment grand pour éviter la cohabitation dans une même couche de fonctions très différentes et suffisamment petite pour éviter que l'architecture ne devienne difficile à maîtriser.

2.1.1 Description des couches

Notre travail ne touche qu'aux couches 2 et 3 du système OSI, mais pour donner un aperçu global au lecteur, voici une description sommaire de chacune des couches qui le composent.

Couche 1: La couche physique

La couche physique s'occupe de la transmission des bits de façon brute sur un circuit de communication. La fiabilité des niveaux binaires des bits y est assurée. La couche physique détermine aussi le niveau de voltage de chaque niveau de bit, la durée des bits, le type de transmission (*duplex* ou *half-duplex*).

Couche 2: La couche Liaison de données

La couche liaison fractionne les données en trames composées de plusieurs octets, transmet ces trames en séquence et génère les trames d'acquittement renvoyées par le récepteur. La couche liaison détermine le début et la fin d'une trame. Elle contrôle

aussi le flux des données de façon à ne pas les perdre. Cette couche s'occupe aussi de la détection des erreurs.

Couche 3: La couche réseau

La couche réseau concerne le transfert de paquets de données à travers un réseau. Ces paquets transitent par un ou plusieurs noeuds intermédiaires avant d'arriver à leur destination. Le niveau réseau prend en charge essentiellement le routage des paquets. C'est également à ce niveau que sont traités les problèmes de congestion et de blocage du réseau.

Couche 4: La couche de Transport

La couche de transport assure le transport de bout en bout de l'information, en débarrassant les niveaux supérieurs de tous les détails des opérations réalisées pour assurer un transport fiable, transparent et efficace des données. Les fonctions réalisées au niveau du transport concernent le multiplexage de bout en bout de connexions de transport sur une connexion de réseau, la segmentation et le groupage des unités de données, ainsi que la régulation de flux de bout en bout. Il fait aussi la traduction entre les adresses de transport et celles de réseau. Il s'occupe de l'établissement et de la fermeture des connexions à travers le réseau.

Couche 5: La couche Session

La couche session assure une série de services qui peuvent se diviser en services de gestion de la session et services de gestion de dialogue. Une session est une connexion entre les utilisateurs. Les services de gestion de la session établissent et terminent la session. Les services de gestion de dialogue assurent la délimitation, le groupement logique et la synchronisation des données.

Couche 6: La couche Présentation

La couche présentation comporte des fonctions qui permettent de modifier ou de traduire les données qui sont échangées. Par exemple, il est possible d'effectuer à ce

niveau des opérations de traduction qui rendent la communication possible entre deux systèmes normalement incompatibles. La couche présentation peut également assurer des fonctions de compression des données ou de chiffrement.

Couche 7: La couche Application

La couche application fournit à l'usager les services qui lui permettent d'exploiter le système téléinformatique. Elle offre, par exemple, des services d'authentification et d'identification des correspondants et elle comporte entre autres des facilités pour synchroniser les programmes qui coopèrent. La couche application comprend également des fonctions de gestion de la couche application et de gestion des systèmes.

2.2 Réseaux Numériques À Intégration de Services (RNIS)

Les applications actuelles en matière de communication font appel à des moyens de plus en plus diversifiés (Stalling 96). Les ancestraux téléphone et télex sont entrain de céder du terrain à de nouveaux services tels que les fax ou les commutations de paquets etc. Il est évident que ces nouveaux supports nécessitent un remaniement profond des moyens de raccordement, des protocoles d'accès, et des différentes interfaces. Le nombre d'interfaces a proliféré de manière dramatique, et une réglementation s'impose: le RNIS a donc pour rôle de permettre à l'usager d'accéder aux différents services de communication à travers un nombre réduit d'interfaces et de protocoles d'accès (stalling). La principale qualité du RNIS réside dans son recours à la technologie numérique le sur le réseau et aux interfaces. Nous exposerons dans ce qui suit une description sommaire des principaux aspects de ces réseaux numériques.

2.2.1 Services

Le RNIS vise à procurer une multitude de services de communication à ses abonnés. Toutefois, la gamme de services offerts varie énormément entre une simple mise en relation de deux usagers ou un transfert de données avec un minimum d'exigences de fiabilité de communication et tout un ensemble d'options relatives aux types de signalisation.

Service	Mode	Débit	Possibilités de transfert	Structure
1)	Circuits	64-Kbps	Sans restriction	8-khz structuré
2)	Circuits	64-Kbps	Parole	8-khz structuré
3)	Circuits	64-Kbps	Audio 3.1 KHz	8-khz structuré
4)	Circuits	2 x 64-Kbps	Sans restriction	8-khz structuré
5)	Circuits	384-Kbps	Sans restriction	8-khz structuré
6)	Circuits	1,536-Kbps	Sans restriction	8-khz structuré
7)	Circuits	1,920-Kbps	Sans restriction	8-khz structuré
Services en mode paquets				
8)	Paquets		Circuit virtuel permanent	
9)	Paquets		Sans filière	
10)	Paquets		Signalisation de l'usager	

Tableau 2-1: Catégories de services RNIS déjà définies.

Le moyen de transfert de l'information constitue un service support entre deux usagers. Le service support ne peut être offert qu'aux points de référence S (*session*) et T (*transport*). Le modèle OSI définit, à l'aide de ses premières couches les rôles et attributs des divers moyens de transfert. En conséquence, il comprend à la fois des caractéristiques

d'accès, des caractéristiques de transfert d'information et des caractéristiques d'exploitation (tableau 2-1).

2.2.2 Canaux de transmission

Les besoins de l'usager sont la base sur laquelle est déterminée la capacité du câble de liaison avec le commutateur RNIS ainsi que le nombre de canaux de communication. Les canaux suivants constituent la base de n'importe quel lien d'accès.

- Canal B: 64 Kbps
- Canal D: 16 ou 64 Kbps
- Canal H0: 384 Kbps
- Canal H11: 1,536 Mbps
- Canal H12:1.92 Mbps

Parmi tous ces canaux, le canal B est le canal de base utilisé par l'usager. Il peut supporter trois types de connexions:

1. Commutation par paquet: L'usager est connecté à un noeud de commutation par paquet, et les échanges de données sont faits à travers le protocole X.25 ou Frame Relay.
2. Semi-permanente: Ce type de connexion est établi à l'avance entre les usagers, et il ne nécessite pas un protocole d'établissement. C'est l'équivalent d'une ligne dédiée.
3. Commutation par circuit: une connexion circuit commuté est établie sur demande entre un usager et l'usager d'un autre réseau.

Le canal D peut servir à transporter les informations de signalisation pour contrôler les appels en mode circuit commuté entre les canaux B impliqués. Il peut également servir en mode de commutation par paquet.

Les canaux H (H0, H11, H12) sont dédiés au transport de débits plus élevés.

Une multitude de combinaison peut exister pour ces différents canaux. Les structures les plus connues sont l'accès de base et l'accès primaire. L'accès de base est composé de deux canaux B duplex à 64 Kbps et un canal D duplex à 16 Kbps. Les hiérarchies de transmission digitale ne sont pas uniformisées partout au monde. En effet, l'Europe utilise la structure (30B +D) avec un débit d'accès de 2048 Mbps alors que l'Amérique du nord et le Japon utilisent la structure (23B +D) équivalente à un débit de 1.544 Mbps.

2.3 Frame Relay

Le protocole X.25 a reigné en maître durant de longues années. Ses principales caractéristiques sont:

1. L'envoi des paquets de commande d'appel pour l'établissement et la libération des circuits virtuels, sur le même canal que les paquets de données (signalisation dans la voie).
2. Le multiplexage des circuits virtuels au niveau de la couche 3.
3. Le contrôle de flux et la récupération d'erreur à la fois dans les couches 2, et 3.

Cette multitude d'opérations a comme conséquence directe de générer un surplus considérable. Notons également que ces opérations devaient se répéter au niveau de

chaque noeud intermédiaire du réseau. Cet encombrement visait essentiellement à palier aux différentes anomalies de transmission rencontrées sur les lignes d'antan.

De nos jours, et avec l'avènement de moyens de communication plus fiables comme la fibre optique, ce surplus est devenu injustifié, voire même encombrant dans la mesure où il ralentit le débit des communications.

L'apport principal du protocole Frame Relay par rapport à ses prédécesseurs est constitué par les points suivants:

- Les messages de commande d'appel sont envoyés sur une connexion logique différente de celle destinée à la transmission des données (une signalisation hors bande). Ce qui veut dire que les noeuds intermédiaires n'ont pas besoin de maintenir des tables d'états relatives aux messages de contrôle.

- La commutation et le multiplexage des connexions logiques se font uniquement au niveau de la couche 2, réduisant ainsi le traitement au niveau de la couche 3.

- Les noeuds intermédiaires n'exercent pas de contrôle de flux ou d'erreurs, ce dernier se fait de bout en bout par les couches supérieures (au cas ou elles sont utilisées).

2.3.1 Architecture

Le protocole Frame Relay, que nous abrégerons dans ce qui suit par Frame Relay se base sur deux plans d'opération: un plan de contrôle (C), qui contient les mécanismes d'établissement et de libération des connexions logiques; et le plan usager (U), qui est responsable du transfert de données entre les abonnés. En conséquence, les protocoles respectifs seront différents: les premiers agiront entre le réseau et l'abonné alors que les

seconds offrent les fonctionnalités de bout en bout. Le protocole du plan-U utilise la recommandation Q.922 pour le transfert des données (fig. 2-3). Cette recommandation est une version améliorée du LAPD (Link Acces Protocol, D-Channel, I.441/Q.921). Seul le noyau de Q.922 (aspects centraux) est utilisé pour le Frame Relay; il contient les fonctionnalités suivantes:

- La délimitation, l'alignement et la transparence des trames.
- Le multiplexage et le démultiplexage de trames en utilisant le champ adresse.
- L'inspection de la trame pour vérifier qu'elle se compose d'un nombre entier d'octets avant l'insertion du zéro-bit ou après son extraction
- La vérification que la trame n'est ni très courte ni très longue.
- La détection des erreurs de transmission.
- Le contrôle de congestion

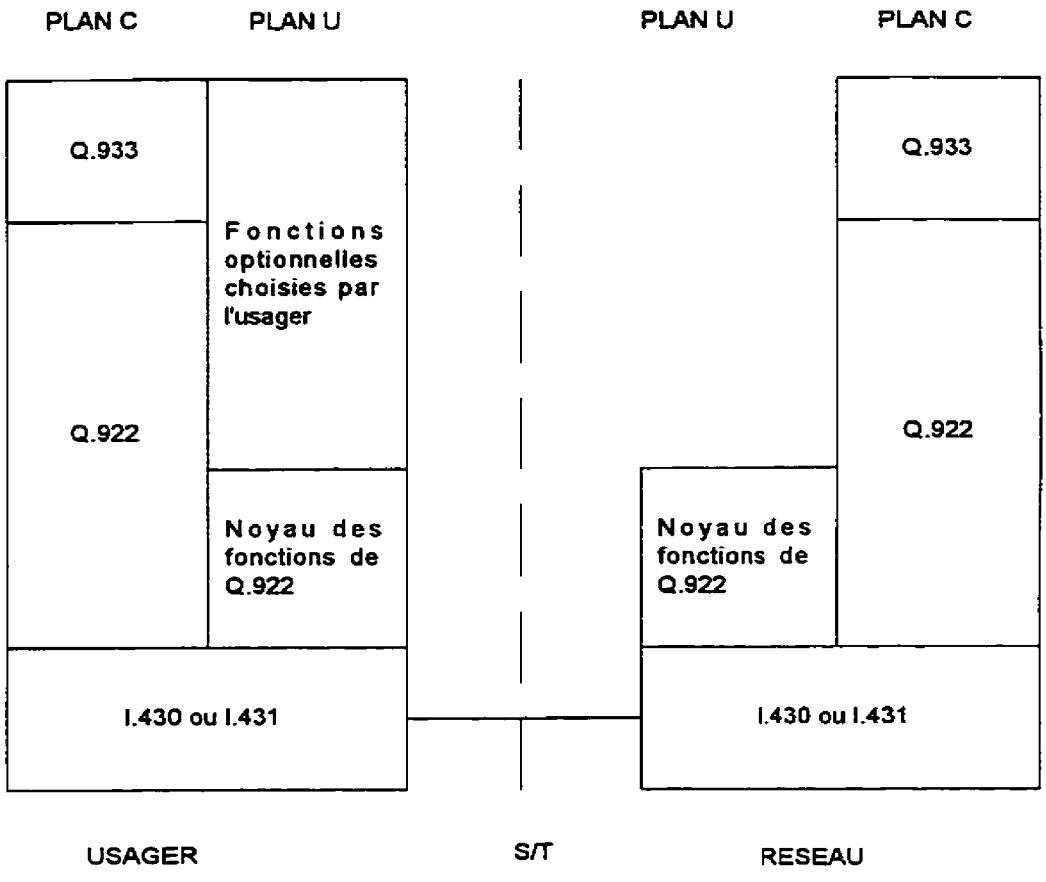


Figure 2-3: Architecture de protocoles de Frame Relay

Le noyau de fonctions de la norme Q.922 n'offre aucun contrôle de flux ou d'erreurs durant le service de transfert des trames, comme montré à la fig. 2-4.

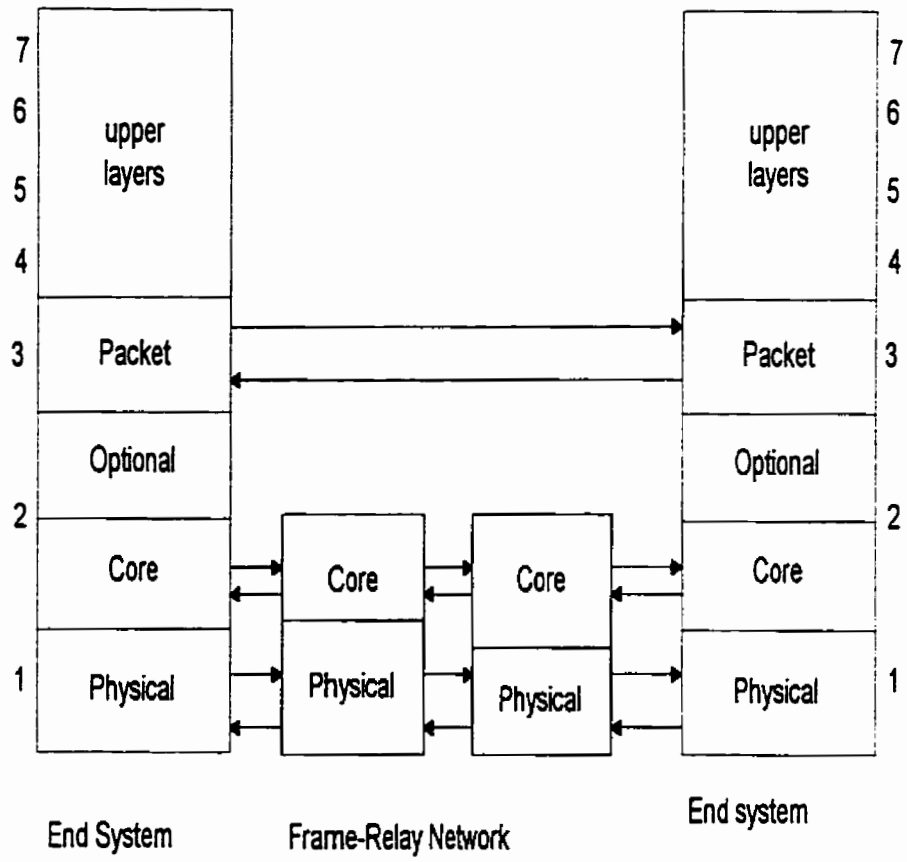


Figure 2-4: Réseau relatif au modèle ISO.

La figure (2-5) montre que cette architecture allège au maximum la quantité de travail effectué par le réseau. Les traitements intermédiaires ne font aucun traitement sur les trames de données qui sont transmises virtuellement. Les couches supérieures auront pour tâche d'éliminer les trames défectueuses.

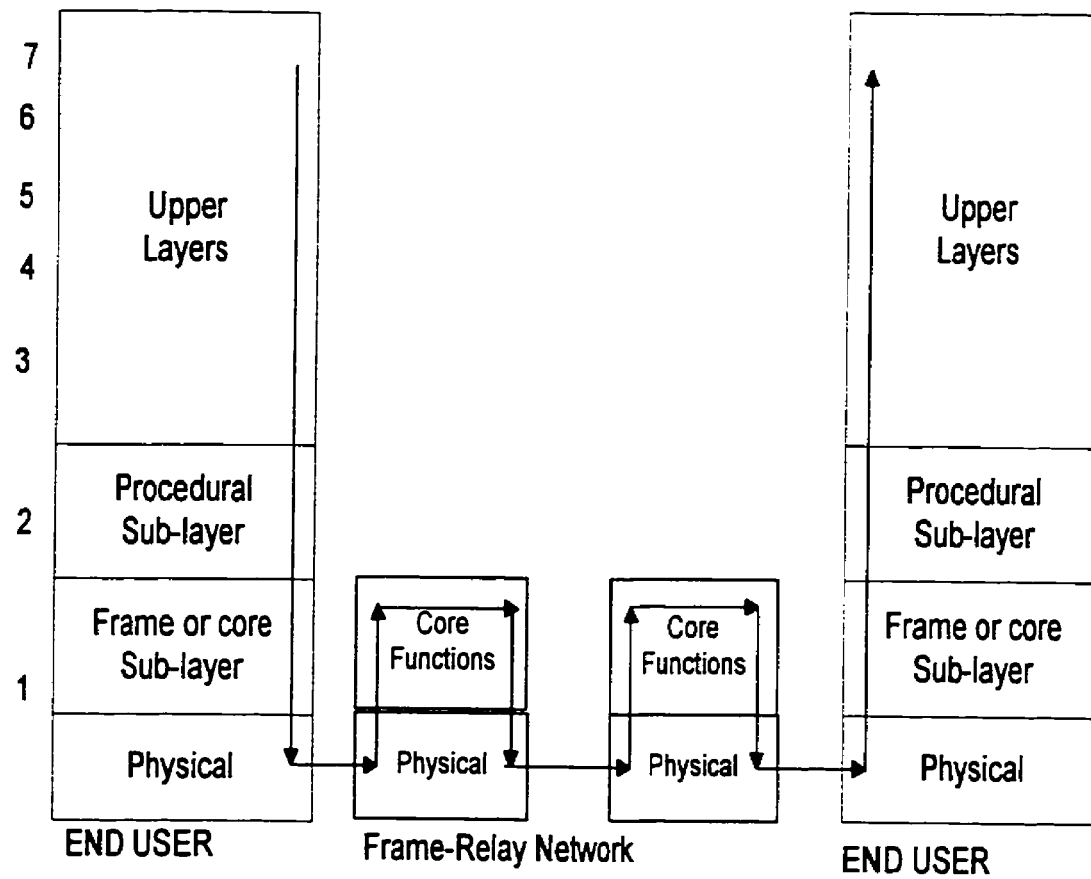


Figure 2-5:Implémentation du service Frame Relay.

2.3.2 Connexion en mode trame

La connexion en mode trame recourt à un nombre défini de signaux (tableau. 2-2). Ces signaux servent à établir les différentes connexions entre des noeuds successifs. Ils sont brièvement décrits dans ce qui suit:

a) Message d'établissement d'appel
1 - Alerte (Alert)
2 - Appel en cours (Call Proceeding)
3 - Connexion (Connect)
4-Accusé de réception de connexion (Connect Acknowledge)
5 - Progression (Progress)
6 - Établissement (Setup)
b) Messages de libération de l'appel
7 - Déconnexion (Disconnect)
8 - Libération (Release)
9 - Fin de libération (Release Complete)
c) Messages divers
10 - État (Status)
11 - Demande d'état (Status Enquiry)

Tableau 2-2: Messages pour la commande des connexions en mode trame

1 - Alerte (*Alert*)**:** Ce message est envoyé au réseau par le terminal demandé et par le réseau au terminal demandeur pour indiquer que l'alerte de l'usager demandé a été déclenchée.

2 - Appel en cours (*Call Proceeding*)**:** Ce message est envoyé par l'usager appelé au réseau ou par le réseau à l'usager demandeur pour indiquer l'initiation de l'établissement de l'appel demandé, et pour indiquer qu'aucune nouvelle information d'établissement d'appel n'est plus acceptée.

3 - Connexion (*Connect*)**:** Ce message est envoyé au réseau par l'usager demandé et à l'usager demandeur par le réseau pour signaler que l'usager demandé accepte la communication.

4 - Accusé de réception de connexion (*Connect Acknowledge*)**:** Ce message est envoyé par le réseau à l'usager demandé pour indiquer que l'appel a été attribué à cet usager. Il peut aussi être envoyé par l'usager demandeur au réseau pour permettre l'application de procédures de commande d'appel symétrique.

5 - Progression (*Progress*)**:** Ce message est envoyé par l'usager ou par le réseau pour indiquer la progression d'un appel en cas d'interfonctionnement.

6 - Établissement (*Setup*)**:** Ce message est envoyé par l'usager demandeur au réseau et par le réseau à l'usager demandé pour déclencher l'établissement d'un appel en mode trame.

7 - Déconnexion (*Disconnect*)**:** Ce message est envoyé par l'usager pour demander au réseau de libérer la connexion en mode trame, ou par le réseau pour indiquer que la connexion en mode trame est libérée.

8 - Libération (*Release*)**:** Ce message est envoyé par l'usager ou par le réseau pour signaler que l'équipement qui émet ce message a déconnecté la connexion en mode trame et qu'il va libérer l'identificateur de connexion de liaison de données (le cas échéant) et la référence d'appel, et pour signaler que l'équipement récepteur doit libérer l'identificateur de connexion de liaison de données et se préparer à libérer la référence d'appel après avoir envoyé le message FIN DE LIBÉRATION.

9 - Fin de libération (*Release Complete*)**:** Ce message est envoyé par l'usager ou par le réseau pour signaler que l'équipement qui envoie le message a libéré la référence d'appel et, le cas échéant, le canal, s'il est libéré, peut servir à nouveau. L'équipement de réception doit libérer la référence d'appel.

10 - État (*Status*): Ce message est envoyé par l'usager ou par le réseau en réponse à un message DEMANDE D'ÉTAT, ou à un moment quelconque au cours d'une communication, pour signaler certaines conditions d'erreur.

11 - Demande d'état (*Status Enquiry*): Ce message est envoyé par l'usager ou le réseau à un instant quelconque, pour demander un message ÉTAT à l'entité homologue de la couche 3. Il est obligatoire d'envoyer un message ÉTAT en réponse à un message DEMANDE ÉTAT.

2.3.3 Composition d'une trame

Une trame est composée des champs suivants (fig. 2-7):

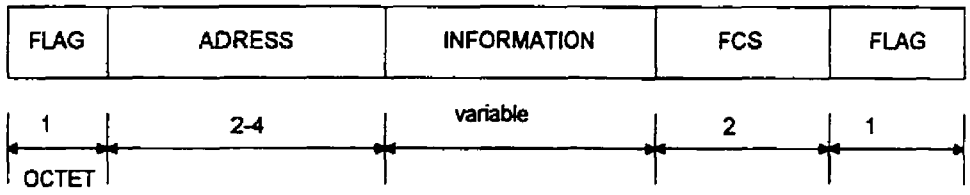


Figure 2-6: composition d'une trame

Le champ *information* constitue la partie utile du signal. Toutefois, son émission doit être précédée d'un octet, *Flag*, qui annonce le début d'une trame. Les champs 2 et 3 sont consacrés à l'adresse. La figure 2-8 montre que ce dernier champ peut aller jusqu'à 4 octets pour certains cas. L'information quant à elle peut être exprimée en diverses longueurs de trames. Dans ce projet, ce champ sera toujours égal à 128 octets. L'avant dernier champ est celui du *FCS (Field Checker Sequence)*, il a été inclus dans la trame bien qu'aucune vrai vérification du signal ne soit effectuée dans le cadre de ce projet et ce par souci de simplification du problème. Le dernier champ est formé par le *Flag* qui annonce la fin de la trame.

Les bits du EA (adress-field extension) définissent la longueur de l'adresse et donc du DLCI (*digital link control identifier*). Le bit C/R est spécifique aux applications, et il n'est pas utilisé par les standards des protocoles de Frame Relay. Le reste des bits dans l'adresse (FECN,BECN,DE) sont relatifs au contrôle de congestion. Dans ce travail, nous

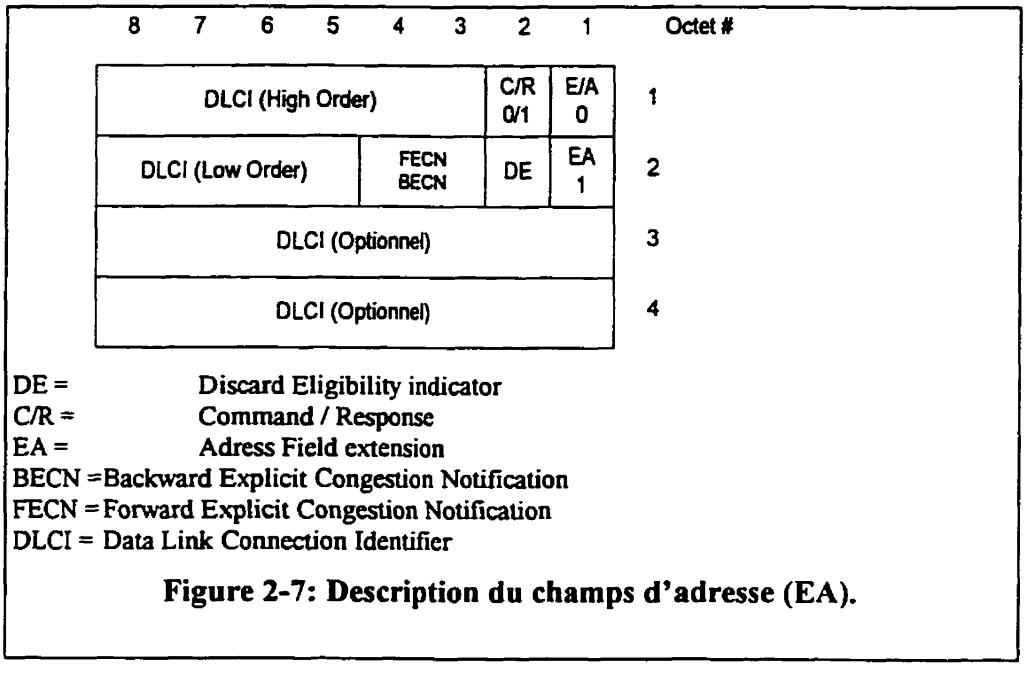


Figure 2-7: Description du champs d'adresse (EA).

n'aborderons pas la partie relative au contrôle de congestion ni à la correction des erreurs car la matière à couvrir sera assez importante et nous fera dévier de notre sujet.

2.4 Introduction aux Réseaux Locaux

Les Réseaux locaux sont ceux qui impliquent l'interconnexion des terminaux, ordinateurs, stations de travail ou tout autres systèmes intelligents à l'intérieur d'un bâtiment ou d'un ensemble de bâtiments constituants un petit campus par exemple. Un

réseau local est généralement limité géographiquement à un intervalle de 0.1 à 10 km. Il opère à des hautes bandes passantes (supérieure à 1 Mbps) et se base sur des supports bon marché (Mouftah et Tavares, 88). Le LAN est probablement le meilleur choix quand il s'agit de combiner divers types de composants et de types de trafic.

Les LANs sont fréquemment caractérisés relativement à leurs topologies. Quatre topologies sont communes: étoile, arbre, boucle et bus.

Réseau en étoile: La grande caractéristique d'un réseau en étoile réside dans l'existence d'un point central de commutation, le moyeu (en anglais *hub*), qui est partagé par toutes les autres stations. Les connexions entre le moyeu et les autres stations se fait à travers des connexions point-à-point. L'élément central utilise la commutation des circuits (*circuit switching*) pour établir un chemin donné entre deux stations qui désirent communiquer (Stalling, 89). L'inconvénient majeur de cette configuration provient de sa grande dépendance de la fiabilité du moyeu.

Réseau en arbre: Un arbre est formé par plusieurs liens qui dirigent les transmissions à partir de chaque station vers un point commun qui constitue le tronc de l'arbre, (*headend)*. Les données sont retransmises du tronc de l'arbre à travers l'arbre vers les points d'interconnexion.

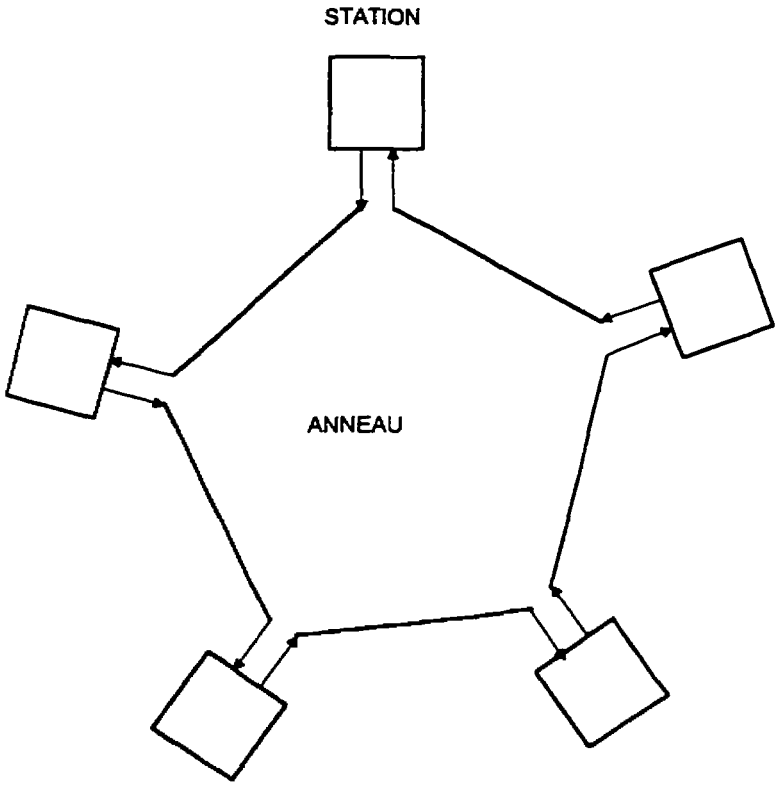


Figure 2-8: Topologie du réseau en anneau.

Réseau en anneau: Un anneau est formé de plusieurs liens point-à-point unidirectionnels dont la connexion est assurée par des répéteurs actifs dans une boucle fermée (fig. 2-9). Chaque station communique avec l'anneau à travers un des répéteurs avec une courte ligne. Une station qui désire communiquer doit absolument attendre son tour pour commencer à transmettre dans la boucle sous forme de paquets. Un protocole de contrôle distribué est utilisé pour déterminer les séquences de transmission de chaque noeud.

Bus: un bus est un long milieu de transmission (fig. 2-10). Les stations sont attachées au milieu de transmission (*medium*) à travers des *Taps*, mieux connus sous le nom anglais de *Bus Interface units* (BIUs). Comme toutes les composantes partagent le même milieu de transmission, seule une paire de composantes peut communiquer à la fois.

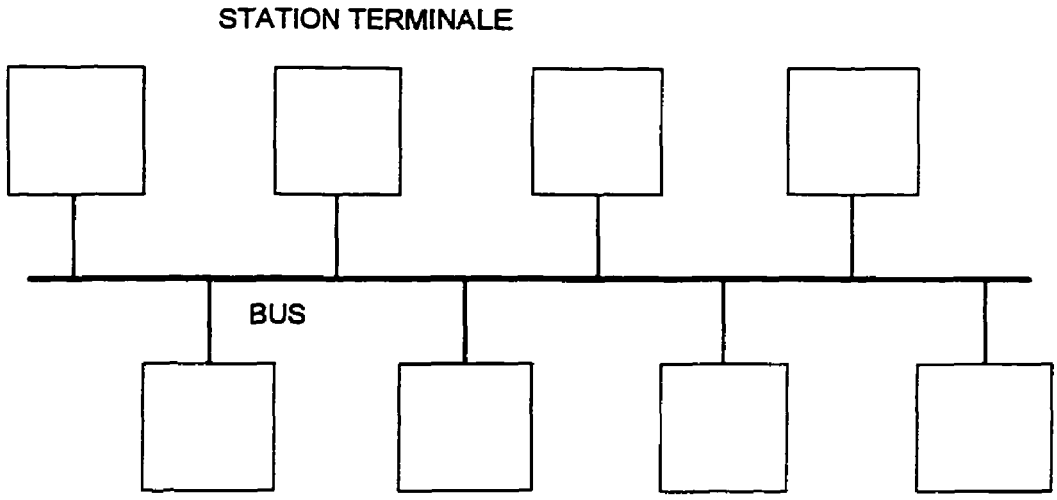


Figure 2-9: Topologie du Bus CSMA/CD.

Toutefois, en pratique, il n'existe que deux topologies de réseaux: le bus de transmission et la boucle à répéteurs actifs. La différence majeure entre les deux est que, dans la première configuration, le signal émis par une station peut être capté par toutes les autres, alors que sur la boucle, la transmission doit être relayée par les répéteurs un à un et ce sur les tronçons concernés de la boucle. Cette différence influe directement sur le choix des protocoles d'acces de Media (nom anglais: *Media acces protocols*) MACs qui peuvent être appliqués à chaque configuration. En plus, et sur le plan de la fiabilité, la boucle s'avère être la configuration la plus vulnérable, et ce paramètre est à considérer lors de toute installation.

2.5 Routage.

Dans les réseaux de télécommunication, le routage est la procédure qui détermine la manière à laquelle un appel est acheminé ou aiguillé. Cette procédure peut considérablement affecter les performances du réseau en termes de coût ou de fiabilité. Les procédures de routage sont composées de trois étapes (Girard, 96). La première implique l'identification des chemins disponibles pour le routage d'un appel particulier. La deuxième implique la sélection du chemin qui sera emprunté par l'appel, ce qui exige la définition d'un certain nombre de lois qui spécifient ce qui arrive à l'appel à son arrivée au réseau. Les réseaux à paquet commuté sont caractérisés par une étape supplémentaire à ce niveau et qui spécifie si un appel doit être routé à tous, même en présence d'un chemin, alors que pour les réseaux à circuit commuté, on suppose que si un chemin est disponible, alors l'appel doit être routé. La troisième étape de la procédure de routage implique la gestion des appels bloqués.

C'est en fonction de sa deuxième étape qu'une procédure de routage est classifiée. Une classe populaire de modèles de routage emploie le routage fixe alterné (*fixed alternate routing).* Cette règle précise un certain nombre d'essais pour router un appel bloqué via un deuxième chemin préétabli, via un troisième, et ainsi de suite. Une autre classe de routage utilise un routage sans dépassement (*overflow*) et avec une règle de partage de charge (*load share rule*). Avec cette règle, à chaque requête de chaque paire origine-destination est assigné un ensemble de chemins relativement à un ensemble de coefficients caractéristiques. Les appels qui n'ont pas reçu de chemin seront définitivement perdus.

Le routage fixe alterné forme une règle simple pour la manipulation des appels à leurs arrivée. Toutefois, il constitue une forme rigide qui ne laisse aucune place à une éventuelle intervention visant à optimiser le routage. La règle 'load share' quant à elle, permet cette intervention en faisant varier les coefficients de routage de chaque chemin.

Dans tous les cas, un algorithme spécifique de calcul du chemin le plus court est adopté à la base. En pratique, les algorithmes de Dijkstra et de Bellman-Ford (De Micheli, 94) sont les plus utilisés.

Le premier algorithme est mieux adapté aux circuits contenant des boucles, mais il n'accepte pas des chemins de pondération négative, et ce contrairement au deuxième. Leurs complexités sont relativement de $O(n^2)$ et $O(n^3)$. Nous reviendrons avec plus de précision sur l'algorithme de Dijkstra dans la section (4.5).

Le présent chapitre a décrit d'une manière assez brève certains concepts relatifs aux réseaux et systèmes de communication. Le chapitre suivant introduira certaines notions de co-design et de synthèse de très haut niveau.

CHAPITRE 3

CO-DESIGN ET SYNTHÈSE DE HAUT NIVEAU

La fidélité de la description d'un protocole de communication est directement proportionnelle au niveau d'abstraction du langage utilisé. Le chapitre précédent nous a présenté le rôle des différentes couches du système OSI dont certaines sont matérielles et les autres logicielles. Le partage M/L implique des choix qui demandent une analyse approfondie. Cette étape vise essentiellement à déterminer les processus qui devront être réalisés en matériel, et ceux qui devront s'exécuter sur un ou plusieurs processeurs.

3.1 Introduction au co-design

Les architectures hétérogènes (M/L) offrent une solution de conception plus efficace du point de vue du rapport prix/performance que ce qui est offert par une architecture complètement matérielle.

Des nombreuses méthodologies ont vu le jour: SpecSyn (Gajski, Vahid et Narayan, 1994), CODES (Buchenrieder, Sedlmeir et Veith 1993), SDW (Antoniazzi et Mastretti, 1990), Gupta et De Micheli (1993) et Ptolemy (Chiodo, Giusto, Jurecska, Lavagno et Hsieh Vincentelli, 1993). Toutes ces méthodes essaient d'intégrer à la fois le matériel et le logiciel dans un même processus de conception. Ces approches diffèrent selon la spécification de l'entrée (modèle de représentation de graphe nommé CDFG: Control Data Flow Graph), la méthode de synthèse (automatique, interactive ou manuelle) et l'architecture cible considérée (monoprocesseur ou multiprocesseur).

L'approche considérée dans le projet COSYMA utilise en entrée une spécification écrite en langage C^x, une extension du C supportant des concurrences au niveau tâches et des contraintes temporelles. Une telle spécification est traduite en une représentation interne sur laquelle une simulation et un profilage préliminaires sont opérés. L'environnement fournit une étape de partitionnement automatique basée sur un algorithme de recuit simulé (simulated annealing algorithm). La stratégie générale considère une solution initiale complètement logicielle et, en analysant les résultats obtenus du profilage, déplace des objets fonctionnels du logiciel au matériel. En outre, un code C est généré à partir des éléments logiciels.

Une autre approche de partitionnement M/L est présentée en (Gupta, 1993). Les étapes de commencement et de la fin sont identiques à COSYMA. La différence principale réside dans la stratégie adoptée par le partitionneur (Vulcan II). Prenant comme départ le graphe système, deux jeux de graphes à limite matérielle ou logicielle respectivement (Hardware-bound et Software-bound) sont générés. Dans la solution initiale, la majorité de la conception vise une solution matérielle alors que les seules fonctionnalités de la partition logicielle sont relatives aux opérations caractérisées par des retards non-déterministes (i.e. primitives de synchronisation et boucles dépendantes de données). Un processus itératif déplace les opérations entre les partitions avec le but de réduire le surplus dû aux communications tout en satisfaisant les contraintes temporelles, le taux d'utilisation du bus/processeur et les contraintes de faisabilité.

Des environnements de co-design qui n'insistent pas sur l'automatisation de l'étape du partitionnement M/L sont aussi proposés. En (Chiodo, Giusto, Jurecska, Lavagno et Hsieh Vincentelli, 1993) les spécifications du système sont modélisées par une machine à états non-finis asynchrone (CFSM) obtenue à partir d'une description VHDL ou ESTEREL. La représentation interne de la CFSM (SHIFT) est adéquate pour l'analyse préliminaire par des techniques de vérification formelles. Chaque module SHIFT est assigné manuellement à une implémentation matérielle ou logicielle et, conséquemment,

traduit en une machine à état (FSM. de Finite State machine) synchrone déterministe. Une telle fonction est activée par les événements associés à la FSM. Elle calcule le prochain état et produit les événements de sortie. Une autre alternative pour l'allocation M/L est présentée par le projet CASTLE (Steinhausen, Camposana et al, 1993). Les systèmes sont modélisés par des langages standards comme le VHDL, Verilog ou le C. La représentation interne (de point de vue logicielle) est hiérarchique et est composée de graphes de flux de contrôle CFG (Control Flow Graph) et de blocs de base. Les transformations de haut niveau, telles que les analyses de durée de vie ou le déroulement de boucles peuvent également être faites. L'approche CASTLE est basée sur le concept de librairie de composantes complexes (processeurs, mémoires, ASICs, etc.) et sur une stratégie library-driven mapping. Quant à l'étape de la synthèse matérielle, un algorithme d'ordonnancement a été développé, il est suivi d'un processus d'allocation de ressources.

Finalement, quelques résultats qui ne concernent pas directement le co-design en entier, mais qui focalisent sur certains aspects spécifiques du partitionnement et sur l'allocation M/L ont été présentés dans d'autres travaux.

En (Barros et Rosentiel, 1992), une méthodologie décrite est basée sur un langage formel (UNITY). Le style de description vise à saisir le calcul parallèle en une manière déclarative en utilisant des variables d'états du modèle et en représentant les transitions à travers des modifications de variables. Avec le langage UNITY, il n'y a pas de concept de contrôle de flux et on peut spécifier autant les comportements synchrones qu'asynchrones. Le processus de partitionnement prend comme départ une analyse qualitative des descriptions menant à une classification des éléments de base de UNITY sur la base de critères pré-définis telles que le synchronisme ou l'asynchronisme, l'exclusion mutuelle et la complexité. Un algorithme de répartition en secteurs (clustering) itératif à deux étages est alors appliqué: le dernier exploite les résultats de la classification. Finalement, les secteurs (clusters) sont alloués à une architecture d'implémentation spécifique, basée sur

un concept maître-esclave où un processeur RISC contrôle l'activation d'un ou plusieurs ASICs.

L'outil PARTIF (Ismail, O'brien et Jerraya, 1994) permet à l'utilisateur d'explorer un partitionnement au niveau système et ce en manipulant un modèle de FSM. concurrents (SOLAR). Un ensemble primitif de règles de transformations (déplacement d'états, fusion d'états) et de décomposition (division/coupure de macro-états) a été défini. À chaque étape d'un processus d'exploration itératif, l'utilisateur sélectionne une règle candidate. Cette règle est vérifiée automatiquement quant à sa faisabilité (prenant en considération les coûts en termes de nombre d'états/opérations et de contraintes de conception). Dans le cas où elle est effectivement faisable, elle est alors appliquée pour générer un système modifié qui deviendra la nouvelle entrée pour la prochaine itération.

Une description de l'entrée peut être donnée dans un langage de description du matériel :VHDL, HardwareC (Gupta et De Micheli, 1993), ou un langage de description système SDL (Buchenrieder, Sedlmeier et Veith, 1993) ou SpecCharts (Gajski, 1994). Dans le premier cas, un format intermédiaire CDFG est généralement utilisé par l'outil de synthèse relatif. Dans le deuxième cas, un format intermédiaire de niveau système est exigé. Le domaine d'application considéré par chaque approche dépend étroitement de la puissance de la description du langage de spécification d'entrée considéré.

Il est également visible que le processus de synthèse puisse être décomposé en deux tâches (fig. 3-1):

- Le partitionnement
- La synthèse de la communication ou de l'interface.

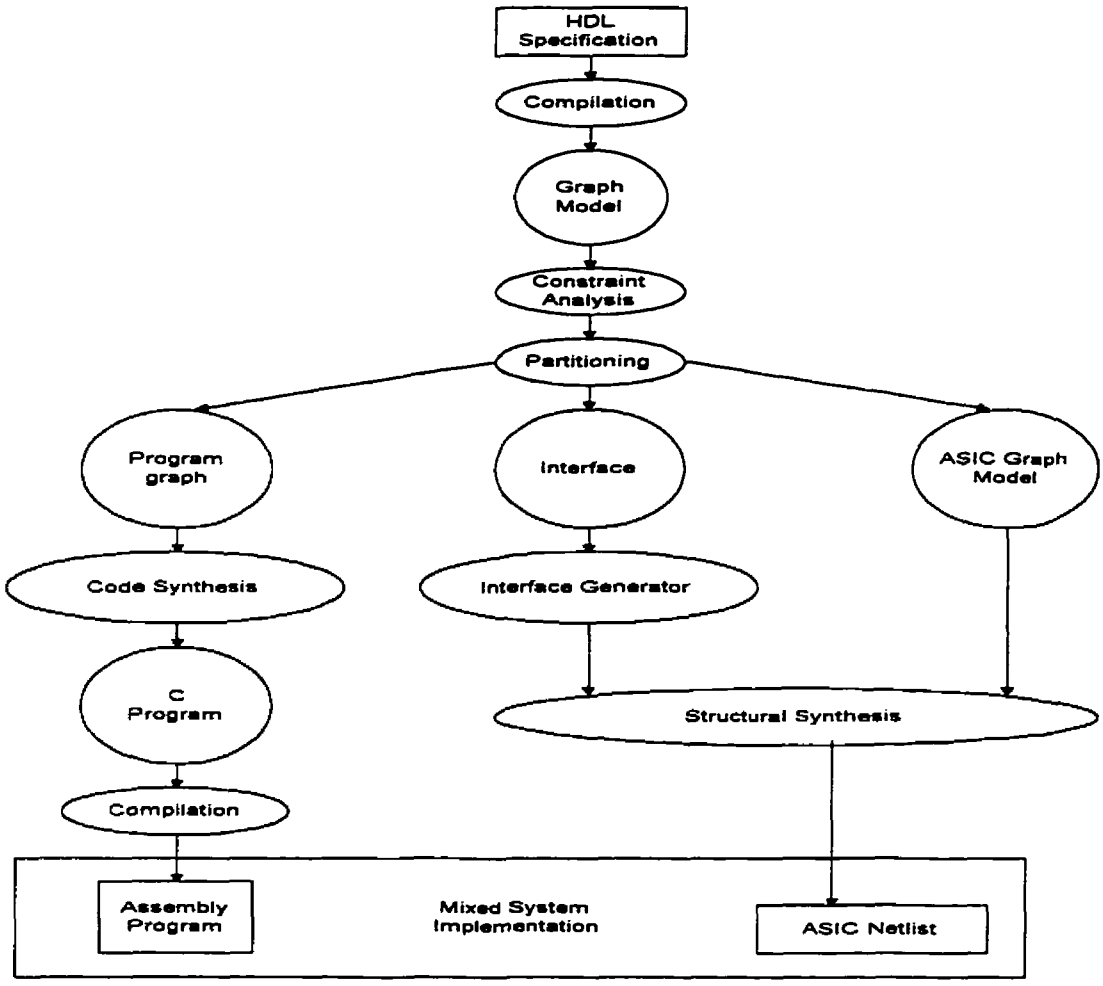


Figure 3-1: Étapes du co-design.

Le partitionnement de la spécification d'un système génère un ensemble de partitions (modules matériels, blocs de communication, modules logiciels) qui seront implémentés selon une architecture cible. La majorité des approches existantes qui partent d'un CDFG utilisent un partitionnement automatique. Néanmoins, vu la complexité sans cesse grandissante, la majorité des systèmes basés sur un modèle de processus communicants utilise l'un ou l'autre du partitionnement interactif ou manuel.

La synthèse de la communication permet la synthèse d'interfaces entre les sous-systèmes. Des approches récentes de co-design sont basées sur l'un des trois modèles de communication:

- Communication point-à-point. '*Fixed communication scheme*' (Buchenrieder, Sedlmeier et Veith, 1993).
- Communication avec un modèle de mémoire partagée (Thomas, Adams et Schmitt, 1993).
- Communication avec un protocole de communication plus ou moins complexe (Gupta, 1993).

L'architecture cible pour implémenter une conception peut être classifiée selon un des trois cas:

1. Architecture multi-puces (Gajski, 1994).
2. Architecture basée sur un processeur et des composantes matérielles (ASICs ou FPGAs) (Gupta, Buchenrieder, Thomas).
3. Une architecture distribuée et flexible (Chiodo et al, 1993). Dans ce dernier cas, plusieurs configurations de processeurs peuvent être utilisés (monoprocesseur, multiprocesseur utilisant une mémoire partagée, architecture basée sur le passage de messages, etc.).

3.2 Une approche de la synthèse de très haut niveau.

La recherche dans le domaine de la conception assistée par ordinateur (CAO) et l'industrie CAO en particulier ont bénéficié d'une évolution exceptionnelle. Comme les

problèmes de la conception au bas niveau d'abstraction sont difficiles à contrôler humainement, et prennent énormément de temps, les concepteurs y ont été confrontés en premier. En conséquence, la recherche en CAO s'est d'abord intéressée aux problèmes de dessin de masques et de *Layout*. Les outils CAO pour la simulation et la synthèse sont arrivés plus tard, en réponse à la complexité des conceptions qui augmentait de façon dramatique pour satisfaire les exigences du marché. Un niveau d'abstraction élevé réduit le nombre d'objets manipulés et considérés par le concepteur, il est donc naturel que la synthèse de haut niveau suive, comme outil, dans la méthodologie de conception des systèmes VLSI.

Un système décrit en composants de plus haut niveau (mémoires, registres, ALUs, Bus, etc.) et spécifié en opérations de plus en plus élevés sur des valeurs de données dans le temps est plus facile à manipuler. En effet, il exprime de façon plus claire la fonctionnalité de la conception, ce qui permet au concepteur de considérer des implémentations alternatives avec plus de facilité. La synthèse de circuits est définie comme étant la traduction d'une description du comportement de ce circuit vers une description de sa structure.

Dans ce travail, nous tentons d'augmenter le niveau d'abstraction de la conception dans tous les domaines, pour en faire une synthèse de très haut niveau. Nous commençons donc par la description du comportement du circuit en langage de haut niveau (VHDL). La deuxième étape consiste à redéfinir le circuit en SDS.

3.2.1 Synthèse de haut niveau

La synthèse du circuit est définie comme étant la traduction d'une description du comportement de ce circuit vers une description de sa structure qui réalise ce comportement. Il y a plusieurs domaines de conception, et la synthèse peut se placer à

n'importe où dans ces domaines: comportemental, structurel ou physique. Dans chaque domaine, des types d'informations différentes sont manipulés, de même que différents aspects du système sont considérés.

3.2.2 Définition de la synthèse de haut niveau

La synthèse de haut niveau est la traduction d'une spécification décrivant le comportement d'un système vers une structure à transfert de registres (TR) réalisant ce comportement. À partir de la spécification à l'entrée, le système de synthèse produit une description de la structure opérative (*Data Path*), qui est un réseau de registres, d'unités fonctionnelles, de multiplexeurs et de bus. La spécification du comportement d'un système peut être formulée par une table de vérité, une équation booléenne ou un programme dans un langage de haut niveau.

3.2.2.1 Hiérarchie de la conception

La description d'un circuit peut être comportementale, structurelle ou physique. Par exemple dans le domaine comportemental, on s'intéresse à la manière dont le circuit agit tout en ignorant celle avec laquelle il est fait.

La hiérarchie de la conception apparaît dans la table (3-1). Cette hiérarchie comprend cinq niveaux.

Niveau système: le système numérique est décrit comme un ensemble de processeurs, mémoires et commutateurs interconnectés.

Niveau algorithmique: les opérations sont exécutées par un processeur individuellement. La séquence des entrées est transformée en une séquence de sorties.

Niveau transfert de registres (TR): Le système est vu comme un ensemble d'éléments de mémoires et de blocs fonctionnels. Le comportement est décrit comme une série de transferts de données et de transformations d'éléments de stockage. La différence entre le niveau TR et le niveau algorithmique réside dans le détail avec lequel est décrite la structure interne.

Niveau logique: le système est décrit comme un réseau de portes et de bascules. Le comportement est spécifié par des équations logiques.

Niveau circuit: le système est vu en termes de transistors dont il est composé.

Niveau *Layout*: Finalement le niveau *Layout* représente la structure interne et le comportement des transistors.

Niveau	Comportement	Structure	Physique
Système	Processus communicants	Processeurs, Mémoires, Commutateurs	Cabinets
Algorithme	Entrée/Sortie	Mémoire, Ports, Processeurs	Cartes, Floorplan
Transfert de registres	Transfert de registres	UALs, Registres, MUX, Bus	Circuits intégrés, Macro-cellules
Logique	Équations logiques	Portes, Bascules	Cellules normalisées, Dessin de masques
Circuit	Équation de réseau	Connexion de transistors	Dessin de masques, Transistors

Tableau 3-1: Hiérarchie de la conception.

3.2.2.2 Utilité de la synthèse de haut niveau

De nos jours, la tendance est d'automatiser la synthèse à des niveaux de plus en plus élevés de la hiérarchie de la conception, pour les nombreux avantages qui en résultent. Parmi les avantages de la synthèse de haut niveau citons:

- Réduction du temps de conception: en automatisant de plus en plus le processus de conception, un circuit peut être conçu et mis sur le marché plus vite. De plus, l'automatisation permet de réduire les coûts de développement;

- Moins d'erreurs: Si le processus de synthèse peut être contrôlé, il y a plus d'assurance que la conception finale corresponde à la spécification initiale. Cela signifie moins d'erreurs et moins de temps de correction des nouveaux circuits intégrés;

- La vérification de la sémantique à un haut niveau de conception: permet de découvrir les erreurs tôt dans le cycle de conception;

- La possibilité d'exploiter l'espace de conception: un bon système de synthèse permet de produire plusieurs conceptions pour une même spécification dans un temps raisonnable. Le concepteur peut explorer différents compromis entre le coût, la vitesse, la puissance, etc;

- Même si à la fin, la conception est produite manuellement, la synthèse automatisée permet de suggérer des compromis intéressants au concepteur;

- Documentation du processus de conception: un système automatisé peut garder la trace des décisions qui ont été prises et pourquoi elles ont été prises et quel est l'effet final de ces décisions;

- Disponibilité de la technologie des circuits intégrés (CI) à un plus grand nombre de personnes: ce genre d'outil fourni la possibilité, à des concepteurs qui ne sont pas forcement des spécialistes, de construire des circuits intégrés qui répondent aux spécifications données. C'est donc un moyen de banaliser la technologie VLSI et de l'ouvrir à un plus grand public.

3.2.3 Concepts et techniques

Le processus de synthèse de haut niveau est composé d'un certain nombre d'étapes qui sont résumées à la figure 3.2. En premier, il faut décrire le comportement du système à concevoir dans un langage de haut niveau tel que Pascal, ADA, etc. ou un langage de description matériel, tel que IPS [Bar92], VHDL comportemental ou DSL etc. Un bon langage de spécification doit contenir un mécanisme pour spécifier les tâches concurrentes. Le système de synthèse doit partitionner la conception en procédures et tâches concurrentes. Cette planification et ce partitionnement sont l'un des problèmes les plus difficiles dans la synthèse de haut niveau: c'est un problème qui doit être considéré à un niveau système.

3.2.3.1 Définition des tâches

Plusieurs tâches de la synthèse sont identifiées, les plus importantes sont expliquées plus loin (fig. 3-2).

1. La compilation: La première étape de la synthèse de haut niveau est habituellement la compilation du langage formel en une représentation interne. Plusieurs approches utilisent la représentation avec des graphes, mais certaines utilisent la représentation avec des arbres. Usuellement, le comportement du circuit est décrit par deux graphes.
    - Graphe de flux de données (*Data Flow Graph*): Représente les différentes opérations et les dépendances entre elles.
    - Graphe de flux de contrôle (*Control Flow Graph*): Montre le séquencement des opérations telles que spécifiées par la description comportementale.

2. L'optimisation de la représentation interne: Cette étape consiste à opérer quelques transformations préliminaires sur la représentation interne. Par exemple l'élimination de sous-expressions communes, l'élimination du code mort, l'expansion de procédures et le déroulement de boucles. L'outil d'optimisation de la représentation interne est identique aux outils d'optimisation du code utilisés dans les compilateurs de langage de programmation.

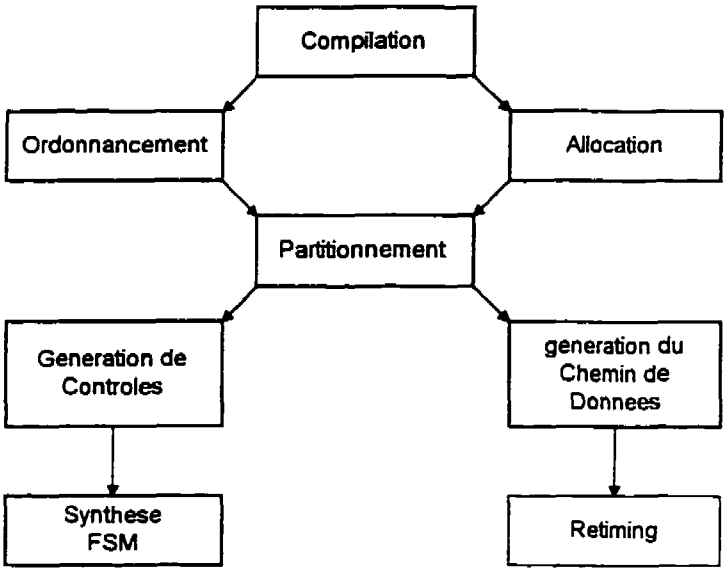


Figure 3-2: Les différentes tâches de la synthèse de haut niveau

Les deux prochaines étapes de la synthèse sont le coeur de la transformation du comportement en structures à transfert de registres.

3. L'ordonnancement: Assigner les opérations à des étages de contrôle pour minimiser une fonction objective donnée tout en respectant les contraintes. Une fonction objective peut inclure, entre autres, le nombre d'étapes de contrôle, le délai, l'alimentation et les ressources matérielles. Le degré de parallélisme de l'exécution de ces opérations est alors défini par l'ordonnancement trouvé. L'ordonnancement est un problème de compromis entre la vitesse de calcul et le matériel.

4. L'allocation: Le problème est de minimiser la quantité de matériel requis pour réaliser le comportement décrit à l'entrée. Le matériel consiste essentiellement en unités fonctionnelles, éléments de mémoire ou registres et réseaux de communication. Les opérations sont assignées aux unités fonctionnelles, les variables aux éléments de mémoire ou registres, et les chemins de données à des interconnexions physiques.

Minimiser tous ces facteurs en même temps est une tâche complexe, par conséquent dans plusieurs systèmes, ils sont minimisés séparément, même si ceci conduit souvent à des résultats sous-optimaux.

Le résultat de ces deux étapes est une structure à transfert de registres, avec des unités arithmétiques et logiques UALs, des registres, des multiplexeurs et des bus.

5. Le partitionnement: Le partitionnement a été introduit dans la synthèse pour déterminer la structure de la conception, en la subdivisant en plusieurs parties qui soient faciles à gérer. Avec des partitions, il est plus facile d'optimiser la conception finale. L'optimisation des partitions résulte en une optimisation du système en entier. Le partitionnement d'architectures procure un moyen de coordonner toutes ces phases en fournissant des informations préliminaires sur la structure du circuit. En effet, il dispose de données physiques, assez tôt dans le processus de conception, sur lesquelles les outils de synthèse peuvent se baser pour leur prise de décision. Donc le partitionnement fournit des informations physiques de performance et de surface. La performance peut être améliorée par le partitionnement à travers la longueur des fils, le parallélisme au niveau opérations et la concurrence au niveau processus. Deux points principaux ont été visés par le partitionnement: la taille de l'architecture relative aux unités physiques et la surface totale de la conception (Lagnese et Thomas, 1989).

Après avoir passé en revue les notions de base relatives au co-design et de synthèse dans le présent chapitre, nous passons au chapitre 4 qui explique les étapes de l'implémentation matérielle et logicielle suivies au cours de ce travail.

CHAPITRE 4

ÉTAPES DE L'IMPLÉMENTATION

Comme il a été expliqué au chapitre 2, un système de communication se compose d'une partie matérielle constituée par la première couche, et d'une partie logicielle constituées par les couches suivantes (figure 4-1). Nous sommes donc appelés à opérer sur deux bases différentes: la première sera logicielle et la deuxième sera matérielle.

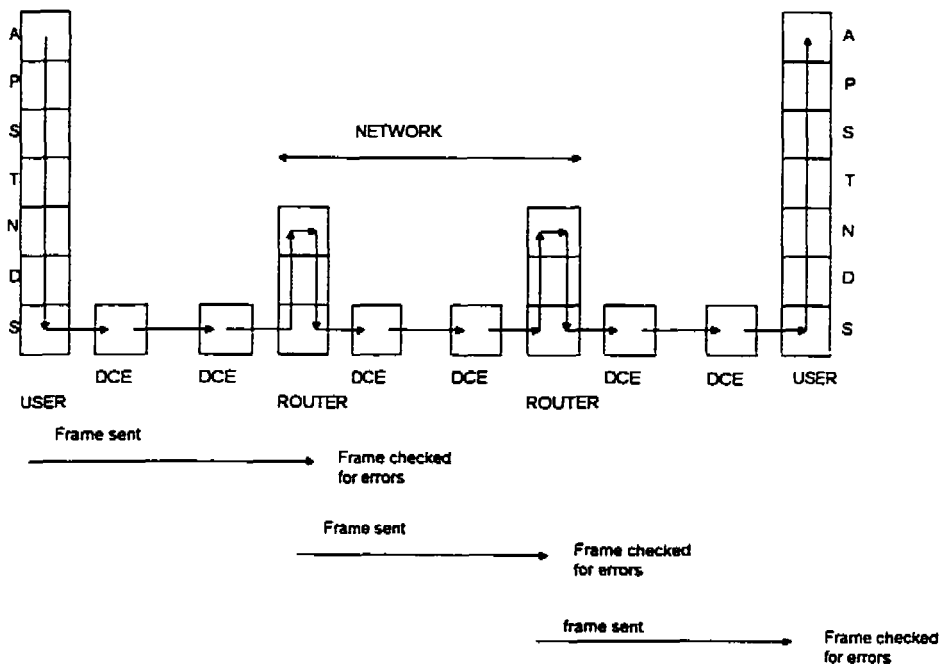


Figure 4-1: Frame Relay dans un réseau.

Considérant ce partitionnement comme initial, la première étape de ce travail consiste à redéfinir un nouveau partitionnement. En effet, nous avons intérêt à maximiser la matérialisation des fonctions contenues dans la deuxième et troisième couches de Frame Relay. Cette matérialisation nous apparaît comme le seul moyen d'accélérer le débit de la communication en raison de la lenteur relative du logiciel comparé au matériel. Les résultats annoncés au chapitre 5 montreront la validité de ce raisonnement.

Le diagramme synoptique de la figure 4.2 montre les trois grands blocs formant notre système, soit l'émetteur (sender), l'aiguilleur (router) et le récepteur (receiver). La partie matérielle du router sera appelée le medium. Pour l'exemple spécifique de ce projet (figure 4.2), le medium sera constitué de 4 noeuds A, B, C et D. La partie logicielle quant à elle comprendra l'information relative aux coûts de chaque liaison de notre réseau. Elle se chargera également de chercher le chemin le plus court entre deux points donnés du graphe entre les quels une connexion doit être établie.

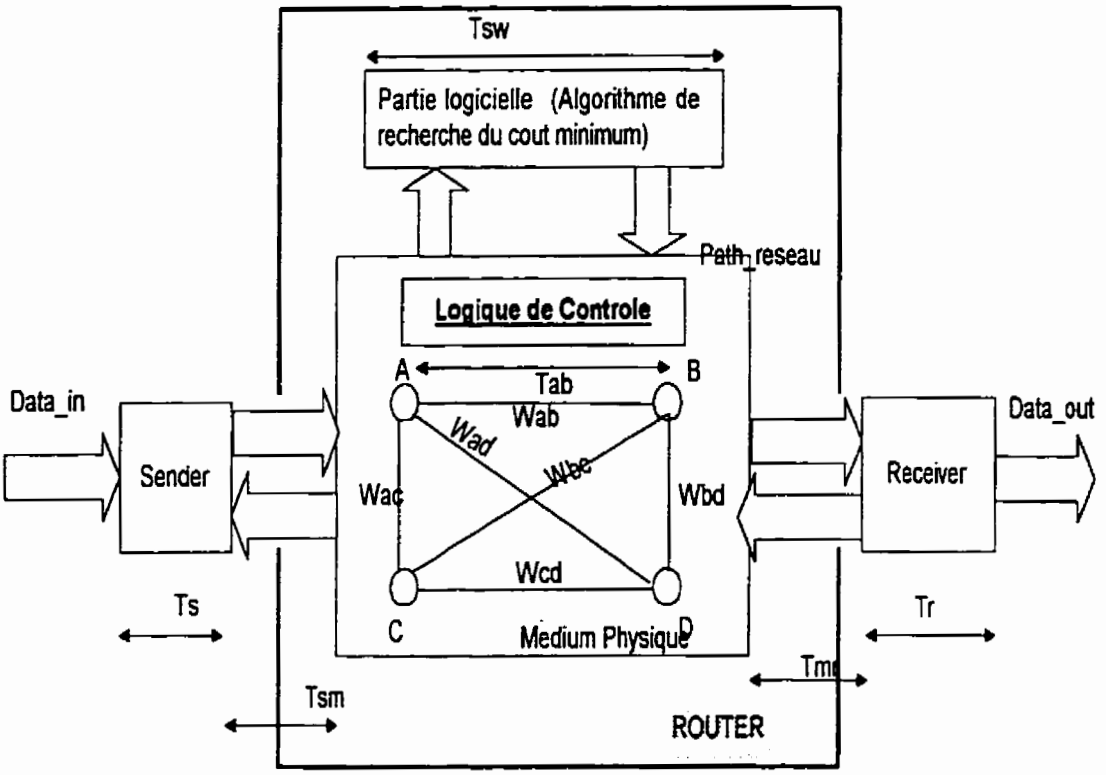


Figure 4-2: Schéma bloc de l'architecture proposée.

Où:

T_S: temps de traitement de l'émetteur.

T_R: temps de traitement du récepteur.

$T_{P(mn)}$: temps de propagation entre les différents blocs (émetteur-medium, medium-récepteur).

T_{SW}: temps de traitement logiciel.

T_{ij}: temps de propagation du signal d'un noeud i à un noeud j. (où $i \neq j$)

T_I: temps consommé par chaque noeud.

En considérant d'abord les métriques: temps d'exécution et temps de communication, représentés respectivement par T_P, T_{ij} et T_T, on a:

- Un temps de propagation: $T_P = \Sigma T_{P(mn)} + \Sigma T_{ij}$ (1)
- Un temps de traitement: $T_T = \Sigma T_{T(i)} + T_{SW}$ (2)

A partir de (1) et (2), le temps total utilisé pour la transmission d'un signal sera:

$$T = T_P + T_T \quad (3)$$

Il est clair que le premier terme varie directement en fonction du choix et de la qualité du support de transmission utilisé (câble coaxial, fibre optique, etc.), en conséquence son amélioration ne relève pas de notre domaine d'étude.

En contre partie, le deuxième terme, en occurrence T_T dépend directement de la sommation des temps consommés par les différents noeuds ainsi que du temps pris par le traitement logiciel. C'est précisément ce paramètre que nous désirons optimiser.

4.1 Hypothèses de départ

Le but de cette recherche est de travailler sur le medium qui constitue un modèle de réseau. C'est sur lui que sera greffée notre approche courante de partitionnement M/L. L'émetteur et le récepteur ne serviront qu'à stimuler le bloc principal qu'est le réseau en question. Par conséquent, pour simplifier notre expérimentation, les diverses options visant à offrir certains services ont été supprimées. Seules les fonctions comprises dans le noyau du fonctionnement du Frame Relay ont été gardées. Cette simplification du modèle nous permettra quand même de généraliser les résultats obtenus pour un protocole intégral. Les champs supprimés ne concernent que des champs relatifs à la vérification des trames au niveau de la réception ou à la signalisation. Leur impact sur le débit de la communication est insignifiant.

De plus, par souci de simplification, ce modèle ne comprend qu'un seul émetteur et un seul récepteur. Toutefois, nous avons prévu aussi bien pour la partie logicielle que la partie matérielle une possibilité d'extension du nombre d'E/S .

D'autre part, sur les réseaux actuels opérants sous Frame Relay, la principale raison de retard est causée par le fait que pour chaque trame transmise sur le réseau, la partie logicielle doit être exécutée. Cette configuration pourrait être justifiable pour les réseaux à grande surface WAN (Wide Area Networks). Le cas est différent pour des applications de haute performance, et avec un support de haut rendement (pas d'erreurs de transmission), ce qui est le cas d'un réseau local, avec un nombre relativement restreint de stations de travail WS (Working Stations). En conséquence, le passage de la trame dans la partie logicielle ne s'effectue que pour la première trame, les trames suivantes suivront le chemin permanent virtuel PVC (Permanent Virtual Circuit) qui a été calculé par la partie logicielle. Il est évident que ce choix comportera des avantages et inconvénients qu'on peut énumérer comme suit:

Avantages

- Réduction du temps de consultation du logiciel. Cette réduction affectera le paramètre T_{ti}, (temps de traitement pour chaque noeud i du graphe).
- Augmentation du débit de transmission sur le réseau, sachant qu'un plus grand nombre de trames peuvent être envoyées dans le même intervalle de temps.

Inconvénients

- Impossibilité d'avoir une allocation dynamique de chemin. En effet, on n'est plus en mesure de modifier le chemin une fois le PVC établi.
- En cas de rupture de liaison, on n'a pas de chemin de secours qui pourra continuer la propagation de l'information.

Il est évident que ce choix constitue un compromis performance/fiabilité dont la justification dépend des conditions d'utilisation et de la qualité de service désirée.

4.2 Approche de co-design considérée

La section suivante expose en détail:

1. La logique qui a dicté notre choix de partitionnement.
2. La méthodologie de design de la partie logicielle.
3. La méthodologie de design de la partie matérielle.
4. Le fonctionnement de l'ensemble.

Rappelons que notre résultat de partitionnement se situe entre les deux limites extrêmes de l'implémentation de protocole:

- l'approche complètement matérielle (Assi, 1994).
- l'approche existante où tout est logiciel, sauf évidemment le support physique

Une réalisation complète en matériel implique la synthèse matérielle d'un algorithme de routage (figure 4-3). Ceci représente une tâche non souhaitable étant donné les raisons suivantes:

- le grand nombre de fonctions mathématiques à réaliser (nous y reviendrons à la section suivante).
- la rigidité du matériel qui ne permet pas une variation considérable des paramètres d'entrée (poids des arcs et nombre de noeuds variables).
- La difficulté de changer les paramètres du réseau qui sont loin d'être stables (ajout d'une station de travail, ajout ou suppression d'un arc, etc.).

En contre partie, nous avons choisi d'implémenter matériellement les noeuds, ainsi que la logique de contrôle (ensemble de fonctions combinatoires) qui les accompagne. Cette dernière partie se chargera d'établir physiquement le chemin dicté par le logiciel (voir section 4.4).

4.3 Implémentation logicielle de l'algorithme de routage

Dans un système de communication, plusieurs algorithmes de routages peuvent servir au calcul du chemin le plus court. Les deux algorithmes les plus répandus sont ceux de Bellman-Ford et de Dijkstra (De Micheli, 1994). Cette section en donne un aperçu assez rapide.

Considérons un graphe dont les arcs sont pondérés G (V,E,W) avec un ensemble de noeuds $V=\{v_i, i=0,1,...,n\}$, le noeud de départ sera appelé v_0. L'ensemble des poids est $W=\{w_{ij}, i,j=0,1,...,n\}$, où $\{S_i, i=0,1,...,n\}$ désigne le poids du chemin le plus court de la source à chaque noeud.

L'équation de Bellman-Ford définit le problème de recherche du chemin le plus court comme:

$$si = \min_{k \neq i}(s_k + w_{ki}); \quad i = 1,2,\ldots,n$$
$$= \min_{k(v_k,v_i)\in E}(s_k + w_{ki}); \quad i = 1,2,\ldots,n$$

Le moyen le plus rapide pour résoudre cette équation est quand le graphe est acyclique. On appelle le fait de trouver une énumération consistante des noeud un tri topologique, il peut être accompli avec une complexité de $\mathbf{O(|V| + |E|) \leq O(n^2)}$.

```
Algorithme de Dijkstra (G(V,E,W)) {
        s0 = 0;
        for (i = 1 to n)
                s_i = w_0,i;
        repeat {
                select an unmarked vertex v_q such that s_q is minimal;
                Mark vq;
                foreach (unmarqued vertex v_i)
                        si = min{si,(s_q + w_q,i)};
                }
                until (all vertices are marked);
}
```

Figure 4-3: Algorithme de Dijkstra

Si le graphe contient des cycles, sa résolution sera alors plus complexe, alors que s'il n'en contient pas, sa résolution peut s'effectuer avec l'algorithme de Dijkstra (figure 4-3). Initialement, tous les noeuds ne sont pas marqués et $\{s_i=w_{0,i}, i=1,2,...,n\}$. Le poids des chemins est alors soit le poids de l'arc qui les relie directement, soit l'infini. L'algorithme commence alors une série d'itérations jusqu'à ce que tous les noeuds soient marqués. Il sélectionne et marque le noeud minimal formant la tête à partir de la source comparée aux autres noeuds dont les têtes ont déjà été marqués. Il doit à chaque fois effectuer des comparaisons entre la valeur trouvée du chemin minimum et les résultats déjà trouvés, pour n'en garder que le minimum des deux. La complexité de cet algorithme est $\mathbf{O(|E| + |V| \log|v|) \leq O(n^2)}$.

Bellman-Ford *(G(V,E,W)){*

$S_0^1 = 0;$

for$(i = 1$ *to* $n)$

$S_i^1 = w_{0,i};$

for$(j = 1$ *to* $n)\{$

for$(i = 1$ *to* $n)\{$

$S_i^{j+1} = \min_{k \neq i}(S_i^j, (S_k^j + w_{k,i}));$

$\}$

if$(S_k^{j+1} == S_i^j \; \forall i)$*return*$(true);$

$\}$

return$(false);$

$\}$

Figure 4-4: Algorithme de Bellman-Ford

Dans le cas général où le graphe contient des cycles et où les signes des arcs peuvent être aussi bien positifs que négatifs, l'algorithme de Bellman-Ford (figure 4-4). peut être utilisé pour détecter la consistance du problème et pour calculer le chemin le plus court recherché. Cet algorithme résout l'équation de Bellman-Ford par relaxation. Il initialise les poids du chemin le plus court aux limites supérieures fournies par les poids des arcs à partir de la source. On doit avoir une convergence du résultat à la fin vers la valeur minimale. Si à l'étape n =|V|-1 la convergence n'est pas obtenue, le problème est alors déclaré comme non consistent.

Dans ce dernier cas, la complexité de l'algorithme est $O(|V| * |E|) \leq O(n^3)$.

Pour le contexte précis de ce travail, le choix s'est porté sur l'algorithme de Dijkstra en raison de sa plus grande facilité de programmation et en raison de

l'absence de tout arc négatif dans le graphe. Les hypothèses qui ont guidé notre programmation étaient les suivants:

- Le réseau contient un nombre de noeuds variant entre 2 et 12.
- Tous les arcs ont des poids positifs.
- Les coûts sont des grandeurs sans dimensions. Ils peuvent exprimer un coût monétaire aussi bien que toute autre métrique spécifique à l'utilisation.
- Les coûts sont générés aléatoirement entre 0 et 20 pour chaque arc. Ces valeurs ne sont qu'à titre indicatif, et toute autre valeur entière positive peut la remplacer.
- Le graphe a la structure d'une matrice (m × n). Cette structure peut très bien s'adapter à un LAN comme c'est notre cas. De plus, toutes les architectures de LANs (section 2.4) peuvent être exprimées par une matrice (m × n).
- Le noeud A constitue la source du graphe. Tous les autres noeuds peuvent servir de puits.
- Sachant que chaque noeud possède une adresse, le programme doit délivrer un vecteur binaire à la sortie dictant la succession de noeuds à suivre depuis la source jusqu'au puits. Ce chemin doit être le chemin le plus court calculé par l'algorithme de Dijkstra.

Structure du programme:

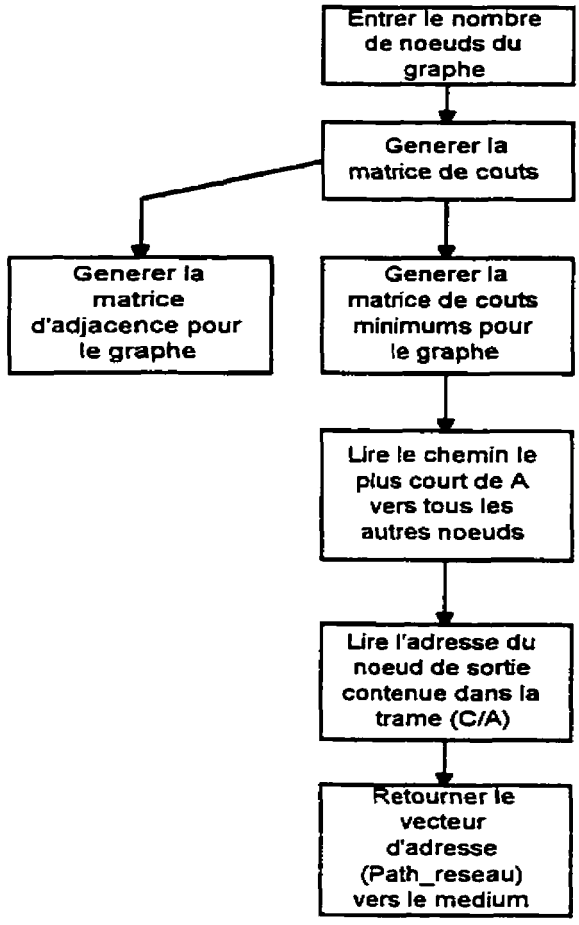


Figure 4-5: Structure du programme de la recherche du plus court chemin.

Ce programme s'est montré très souple à l'utilisation. Il permet un grand nombre de modifications de paramètres du réseau. Sa sortie a été restreinte à une application spécifique pour ce modèle (nombre de noeuds maximum ne dépassant pas 10, avec A comme source), mais rien ne l'empêche d'accepter d'autres conditions de travail. Les tableaux (4-1) montrent un exemple de traitement de l'algorithme pour un graphe de 6 noeuds.

	A	B	C	D	E	F
A	3	3	14	19	0	15
B	11	5	3	14	0	5
C	9	12	14	17	4	12
D	1	3	14	12	0	5
E	13	7	10	16	9	7
F	16	6	18	14	5	3

(a)

	A	B	C	D	E	F
A	0	3	6	16	0	7
B	11	0	3	14	0	5
C	9	11	0	17	4	11
D	1	3	6	0	0	5
E	13	7	10	16	0	7
F	15	6	9	14	5	0

(b)

	A	B	C	D	E	F
A	0	1	1	1	0	1
B	1	0	1	1	0	1
C	1	1	0	1	1	1
D	1	1	1	0	1	1
E	1	1	1	1	0	1
F	1	1	1	1	1	0

(c)

Chemins minimums à partir de A:

A à B: A => B

A à C: A => B => C

A à D: A => E => D

A à E: A => E

A à F: A => E => F

(d)

Tableau 4-1: Simulation de l'algorithme de minimisation de coûts avec n = 6

(a): Matrice des coûts initiaux du graphe, (b): Matrice des coûts minimaux du graphe,
(c): Matrice d'adjacence du graphe, (d): Chemins minimums à partir de A.

Signalons que cette souplesse d'utilisation serait très difficile, voir même impossible à obtenir avec un design matériel de cet algorithme.

Le vecteur de sortie, appelé path_reseau sera mémorisé dans le registre tampon (buffer) portant le même nom situé dans le medium. Le module logiciel ne sera sollicité de nouveau qu'en cas de rupture de liaison ou de demande d'établissement de nouvelles liaisons.

Les noeuds déjà occupés par la connexion # 1 seront rayés de la liste dressée par la matrice d'adjacence, en d'autres mots, ils ne seront pas présents dans le graphe pour les communications devant être établies avant l'achèvement de la première communication. Cette condition engendrera la création d'une nouvelle matrice de coûts minimums pour les prochaines connexions qui doivent se passer durant l'accomplissement de la première.

Cette solution représente, elle aussi, un compromis qui présente ses avantages et ses inconvénients. Nous pouvons lui reprocher de ne plus permettre une utilisation dynamique de chaque noeud, i.e. situation où le noeud est partagé par plusieurs appels par multiplexage, mais elle offre plus de vitesse d'établissement de liaison entre les différents noeuds, moins de temps perdu donc un meilleur débit. Ces paramètres peuvent être justifiés pour un LAN de petite dimension, mais sont à revoir pour un réseau de moyenne ou grande dimension.

4.4 Implémentation matérielle du Router

La partie matérielle de ce travail est la plus longue et la plus délicate à concevoir. En effet, les données et paramètres du design sont très diversifiés, voir même contradictoires en certains moments. La partie la plus difficile est de déterminer les meilleurs compromis coût/performance. Ensuite il s'agit de rendre cette partie

compatible avec le bloc logiciel déjà conçu. Dans ce qui suit, nous exposons les différentes étapes de notre architecture.

4.4.1 Déroulement d'un appel

Sachant que le réseau comprend un medium à quatre noeuds, le déroulement d'un appel suit les étapes suivantes:

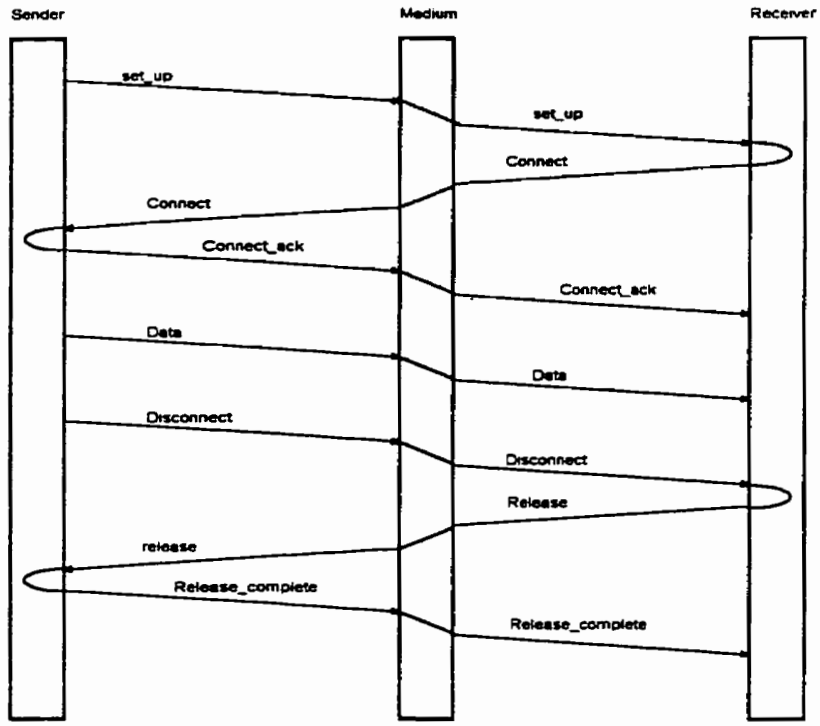


Figure 4-6: Déroulement d'un appel.

L'établissement d'une connexion est toujours précédé de 3 signaux de signalisation: soit le *Set_up*, le *Connect* et le *Connect_ack*. La rupture de la communication est signalée par les signaux *Disconnect*, *Release* et *Release_complete*. Ces signaux de signalisation sont spécifiques au protocole Frame Relay. Leur rôle consiste à signaler aux différents constituants du réseau l'état de leur correspondant et l'état qu'ils doivent eux mêmes prendre (repos, réception ou émission, etc..).

Le tableau (4-2) donne les différentes valeurs par les signaux de signalisation Frame Relay:

Nom du signal	Valeur
Set_up	0000 0101
Connect	0000 0111
Connect_ack	0000 1111
Disconnect	0100 0101
Release	0100 1101
Release_complete	0101 1010

Tableau 4-2: Valeurs des signaux de signalisation

4.4.2 Composition d'une trame

Nous avons vu à la section (2.3.3) la composition d'une trame, la présente section montre la manière à laquelle ces champs ont été définis en VHDL.

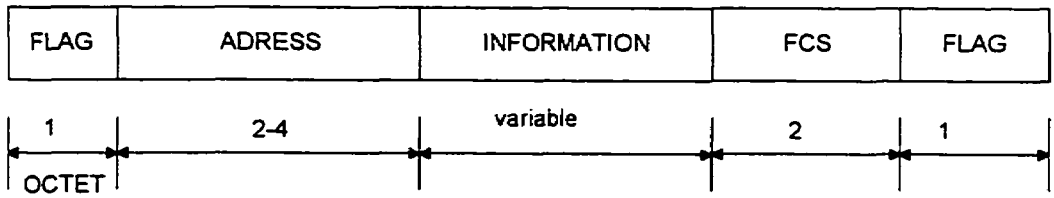


Figure 4-7: Trame de Frame Relay.

Ces différents champs sont générés au niveau de l'émetteur. Le medium doit analyser le champ d'adresse et transmettre cette donnée au bloc logiciel. Ce dernier se chargera de calculer le chemin le plus court en vertu de l'adresse désirée.

Les autres champs seront pris en charge par le récepteur. Ils serviront essentiellement à délimiter les trames d'information et à corriger les éventuelles erreurs de transmission.

Dans le cadre de ce projet, aucune correction n'a été prévue. Cette option constitue à elle seule un grand domaine d'étude qui risquerait de nous faire dévier de notre axe de recherche principal. L'originalité du protocole frame relay réside dans le signalisation des trames erronées. En effet, c'est au niveau du récepteur que les corrections se passent, donc aucun ralentissement de débit n'est observé. Il est donc évident que cette simplification n'a aucun effet sur l'exactitude des performances obtenues par ce travail.

Le tableau (4-3) montre les valeurs des champs de l'entête et de la queue de la trame.

Nom du signal	Valeur
Flag	0111 1110
Adress	0110 0000
FCS	0000 0110

Tableau 4-3: champs de la trame

L'approche considérée pour la réalisation matérielle consiste à définir chaque module séparément, en langage VHDL. Par la suite, il s'agit de concevoir un fichier *Makefile* qui fera fonctionner l'ensemble d'une manière harmonieuse.

4.4.3 Description VHDL

Le VHDL (Very high speed integrated circuit Hardware Description Language) est un langage qui sert à décrire des systèmes matériels à un haut niveau

d'abstraction. Le VHDL permet des descriptions temporelles et hiérarchisées de systèmes ayant des modes de fonctionnement concurrents. Ce langage couvre principalement deux domaines d'application: la modélisation de systèmes en vue de la simulation et la spécification de systèmes en vue de la synthèse d'un circuit intégré (Airiau et al, 1990).

Il existe plusieurs catégories d'unités de conception:

1 - La spécification d'entité (entity): c'est la description d'un système électronique du point de vue de ses entrées-sorties. Elle se compose de deux parties principales: une interface et une ou plusieurs architectures.

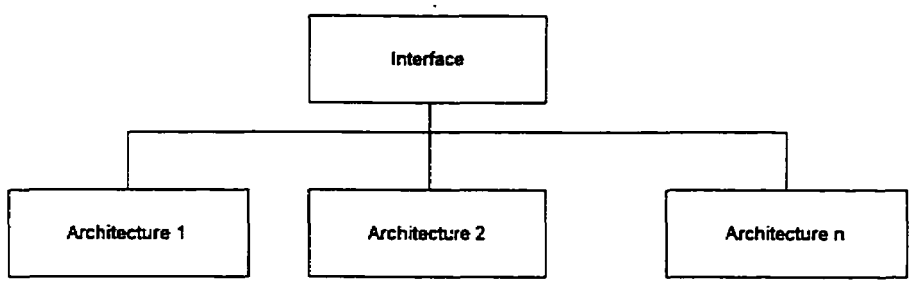


Figure 4-8: Structure globale d'une entité

2 - L'architecture (Architecture): c'est le corps de l'entité. Elle décrit l'architecture interne d'un circuit en utilisant une description structurelle, flux de données, comportementale ou hybride. Dans une architecture, nous retrouvons, comme pour un programme conventionnel, les déclarations de variables et les signaux internes à l'objet, ainsi que les instructions proprement dites (affectations de signaux et de variables, test, etc.).

Dans une architecture, des types de données et des fonctions ou des procédures, regroupés éventuellement à l'intérieur d'une même entité appelé package,

peuvent être utilisés. Les informations contenues dans un package sont en fait transparentes à l'usager qui peut en faire usage au besoin sans connaître leur structures intrinsèques. Dans un package, on ne peut déclarer des variables ni des signaux.

3 - La configuration (Configuration): Cette unité décrit la correspondance entre les composantes déclarées dans les architectures et celles précisées dans les entités.

4.4.3.1 Composantes VHDL

On qualifie de composantes (mot clé component), les éléments constitutifs d'une description VHDL. Une composante repose sur une entité définie préalablement. Elle spécifie le corps à associer à l'interface de l'entité. La création d'une composante porte le nom d'instantiation, où un ou plusieurs processus VHDL générés se comportent selon la description de l'entité les ayant engendrés. Notons qu'une entité peut servir de base à plusieurs instantiations.

4.4.3.2 Processus VHDL

Un processus (mot clé process) en VHDL est une suite d'instructions exécutées de façon séquentielle. Un processus se déroule parallèlement aux autres processus présents dans le système. Chaque processus est sensible à une liste de signaux. Dès que l'un d'eux est modifié, les processus qu'il influence sont réveillés, exécutés et ensuite replongés dans leur état de suspension.

4.4.4 Description VHDL de la partie matérielle du modèle

La description VHDL a pris comme point de départ la répartition de l'ensemble en trois grands blocs, à savoir l'émetteur, le medium et le récepteur. Le medium est composé d'un nombre de composantes (components) exprimés dans d'autres sous programmes pour des raisons de facilité d'expression. Un programme (Makefile) aura pour tâche de faire rouler l'ensemble dans les conditions établies d'avance.

Un point mérite une clarification ici: les systèmes de communication sont en général asynchrones. Chaque bloc est chargé de recouvrir le signal d'horloge au cours de la transmission. Toutefois, ce travail ne constitue qu'un modèle de validation de partitionnement M/L. Le choix d'un système synchrone, ici, est justifié par le souci de faciliter les communications à l'intérieur d'une même puce.

Pour cette partie de travail, deux environnements de travail ont été utilisés, ce sont le MentorGraphics et le Synopsys, nous y reviendrons ultérieurement.

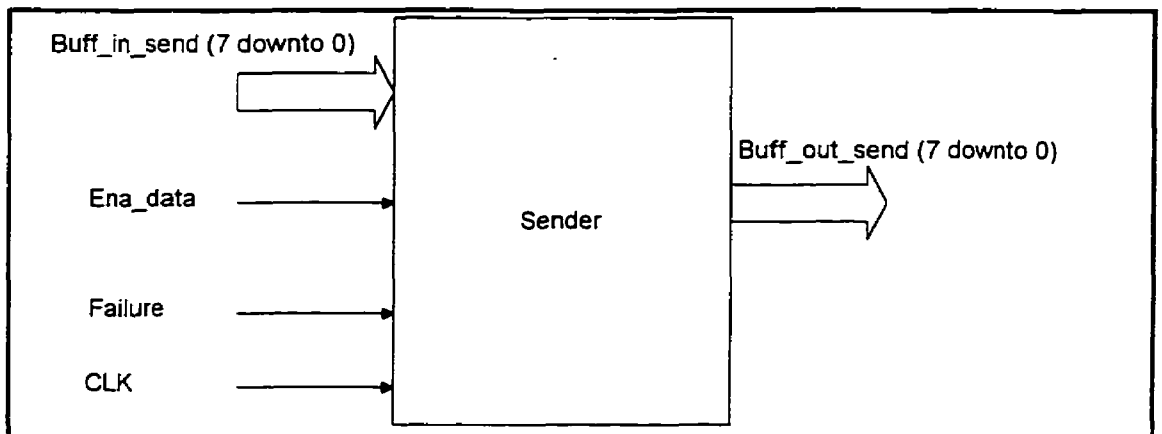
4.4.4.1 Description de l'émetteur

L'émetteur a pour rôle de recevoir les signaux de data_in, et de demander au medium d'établir une éventuelle connexion de l'appel devant être acheminé jusqu'au récepteur (figure 4-8). À la réception du Data_in, et au passage du signal Ena_data de '0' à '1', il doit demander l'établissement d'une liaison avec le medium (set_up) tout en lui spécifiant l'adresse finale de l'émetteur à atteindre. Sur réception du signal de connexion du medium (Connect), il doit répondre par le signal Connect_ack avant de commencer l'envoie des données.

L'émetteur communique avec son environnement à travers les ports suivants:

1. Ena_data: par lequel il reçoit le signal de mise en marche.
2. Failure: signale à l'émetteur la présence d'une anomalie majeure dans le réseau, et lui ordonne de retourner à l'état de repos.

3. CLK: horloge.
4. Buff_in_send: c'est le buffer d'entrée qui permet à l'émetteur de recevoir les divers signaux de contrôle (entête ou queue de trame, signal de signalisation) ou de données. Ce port peut recevoir les signaux issus du medium ou ceux arrivant de l'extérieur et emmenant des données. C'est la partie signalisation qui lui permet de distinguer entre les deux.
5. Buff_out_send: c'est le buffer de sortie qui permet à l'émetteur d'émettre les divers signaux de contrôle (entête ou queue de trame, signal de signalisation) ou de données. Ce port ne peut envoyer les signaux que vers le médium




```
entity FR_SENDER is
        port (  Ena_data            : in std_logic:= '0';
                Failure             : in std_logic:= '0';
                BUFF_IN_SEND        : in std_logic_vector (7 downto 0);
                BUFF_OUT_SEND       : out  std_logic_vector (7 downto 0)        ;
                CLK                 : in std_logic);
end FR_SENDER;
```

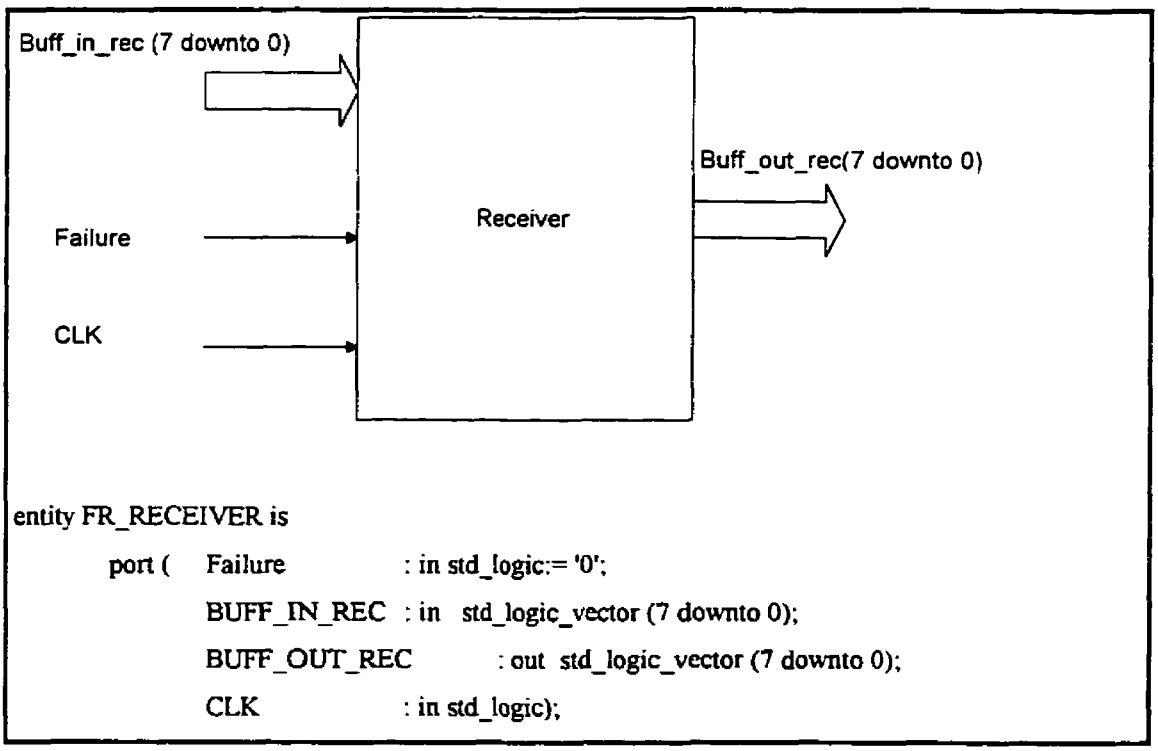
Figure 4-8: Schéma synoptique et entité de l'émetteur.

Les données sont envoyées sous forme de trame de 133 octets. La tête de trame occupe 3 octets, la queue 2 et la plage information 128 octets.

En cas de rupture de connexion ou de fin de trame, l'émetteur envoie le signal Disconnect au medium. Sur réception du signal Release il répond par Release_complete pour indiquer au réseau de retourner à son état de repos.

4.4.4.2 Description du récepteur

Le récepteur a pour rôle de recevoir le signal issu du buffer de sortie du medium, et de le convertir en data_out. Sur réception du signal de demande de connexion set_up avec le medium, il répond par le signal Connect. Il doit recevoir le signal Connect_ack lui signifiant de se mettre prêt à recevoir les données émises par l'émetteur (figure 4-9).



```
end FR_RECEIVER;
```

Figure 4-9: Schéma synoptique et entité du récepteur.

Le récepteur communique avec son environnement à travers les ports suivants:

1. Failure: signale au récepteur la présence d'une anomalie majeure dans le réseau, et lui ordonne de retourner à l'état de repos.
2. CLK: horloge.
3. Buff_in_rec: c'est le buffer d'entrée qui permet au récepteur de recevoir les divers signaux issus du medium, et qui peuvent aussi bien être de contrôle (entête ou queue de trame, signal de signalisation) ou de données.
4. Buff_out_send: c'est le buffer de sortie qui permet au récepteur d'émettre les divers signaux de contrôle (entête ou queue de trame, signal de signalisation) ou de données. Ce port peut soit communiquer avec le medium ou avec l'extérieur pour lui transmettre les signaux de données qu'il reçoit. C'est la partie signalisation qui lui permet de distinguer entre les deux.

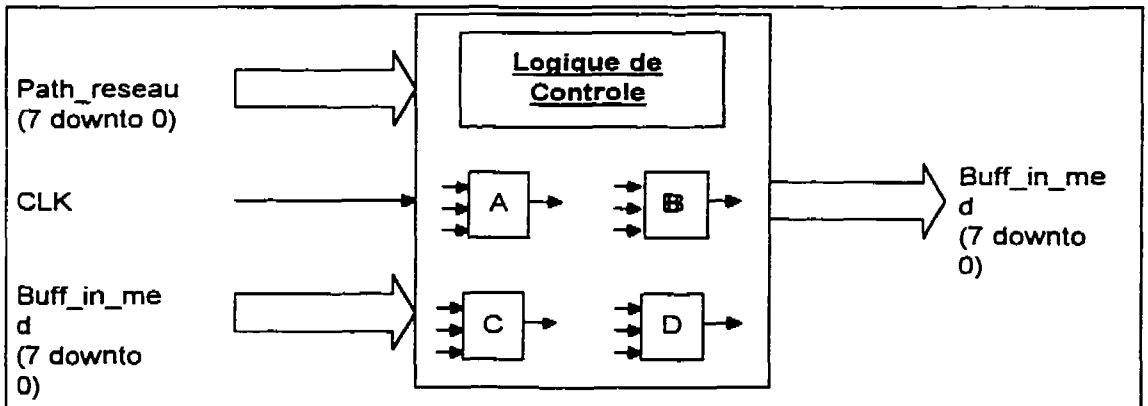
4.4.4.3 Description du medium

Comme expliqué précédemment, c'est au medium que la majeure partie du travail a été consacrée. Sachant qu'en pratique deux types de topologies peuvent exister pour les LANs, à citer le bus et la boucle (section 2.4) la configuration qui a été retenue pour le medium est celle d'un graphe complet à 4 noeuds (tous les noeuds se touchent).

Les noeuds sont contrôlés par le bloc de "logique de contrôle". Le noeud est en fait un commutateur qui bascule le signal d'entrée vers la sortie après un délai préalablement fixé. La durée de ce délai est fonction de la technologie adoptée (figure 4-10).

Le medium communique avec son environnement externe à travers les ports suivants:

- Path_reseau: qui emmagasine la valeur du vecteur d'adresse retourné par le logiciel. Pour ce projet, le *Path_reseau* est formé de 8 bits. Chaques 2 bits constituent l'adresse d'un noeud, de sorte que le *Path_reseau* n'est autre qu'un séquencement de 4 noeuds selon l'ordre retourné par l'algorithme de Dijkstra.
- Buffer_in_med: c'est le buffer d'entrée qui reçoit tous les signaux entrants au medium, qu'ils soient de contrôle ou de données. La séparation des deux types de signaux se fera grâce aux sélecteurs (Sel1, Sel2 ou Sel3) qui comprennent des *FSM*. permettant l'aiguillage des différents signaux.
- Buffer_in_med: c'est le buffer de sortie par lequel tous les signaux de contrôle ou de données doivent forcément transiter. La séparation des signaux se base sur le même concept que pour le *Buffer_in_med.*
- CLK: c'est le port d'entrée des signaux d'horloge.




```
entity fr_Medium is
        port(  path_reseau      : in  std_logic_vector (7 downto 0);
               BUFF_in_MED      : in  std_logic_vector (7 downto 0);
               BUFF_out_MED     : out  std_logic_vector (7 downto 0);
               CLK              : in std_logic);
end fr_Medium ;
architecture RESEAU_ARCH of fr_Medium is
```

```
component NOEUD
        port(  Ctrl_noeud        : in  std_logic;
               Adresse_noeud     : in  std_logic_vector (1 downto 0);
               Data_in_m         : in  std_logic_vector (7 downto 0);
               Data_out_m        : out std_logic_vector (7 downto 0);
               CLK               : in  std_logic);
end component;
...
end RESEAU_ARCH
```

Figure 4-10: Schéma synoptique et entité du medium.

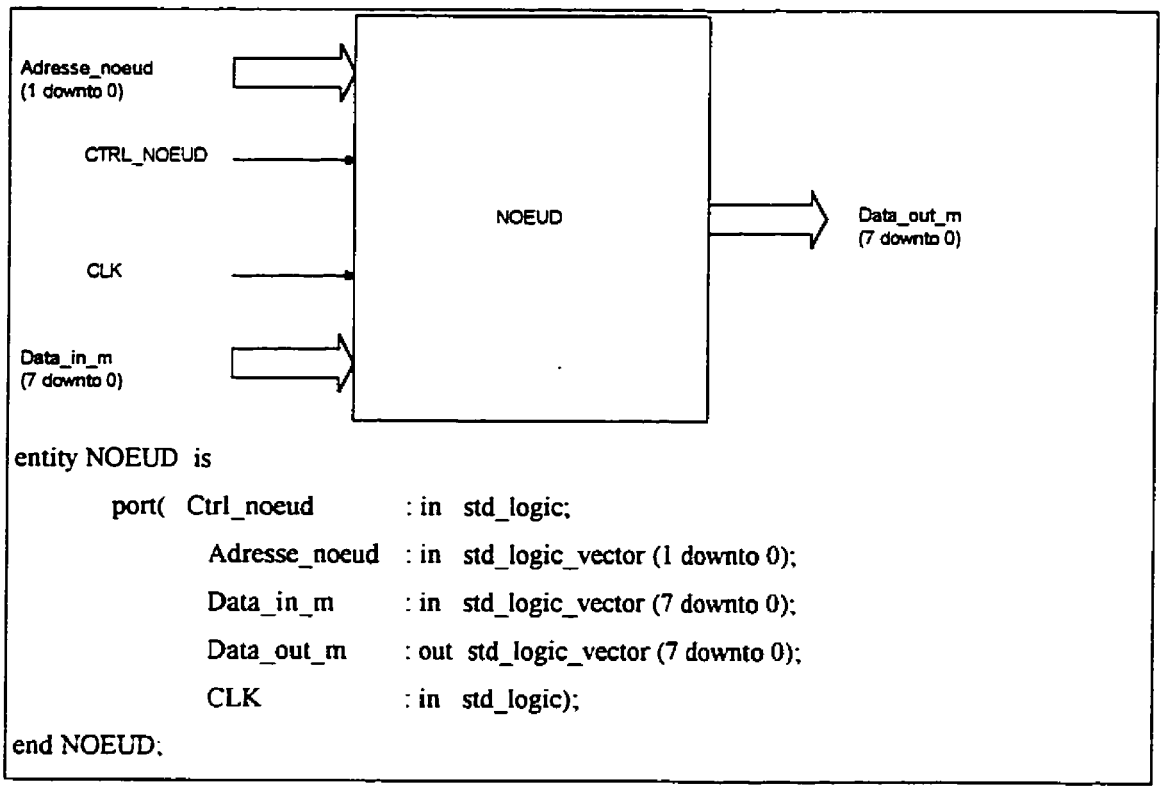
Le medium comprend, en plus des noeuds, la partie logique qui doit:

1. Assigner matériellement le chemin calculé par la partie logicielle. Cette partie est formée par une succession de multiplexeurs.
2. Générer les vecteurs de signalisation Frame Relay (section 4-6-1).
3. Gérer le déroulement d'un appel et l'état du réseau.

Le choix de multiplexeurs en cascade servant à l'implémentation du chemin a été pris après le passage en revue des différentes solutions possibles. Les noeuds ont été triés sous forme d'arbre de choix. Cette architecture peut sembler être assez dispendieuse en surface pour des LAN de moyennes ou grandes dimensions, mais la pratique reflète l'existence de deux topologies réelles (bus et boucles), et que toutes les autres topologies en sont dérivées. Ce fait est un argument de taille pour le choix de cette architecture, car nous distinguons deux cas:

- cas d'un bus: le bus 'voit' tous les noeuds en même temps. En conséquence, un seul niveau de multiplexeurs est franchi pour acheminer les signaux de l'entrée du medium vers sa sortie, et la complexité d'une telle implémentation est linéaire.
- Cas d'une boucle: tous les noeuds sont en cascades. Le signal arrivant à l'entrée du medium sera véhiculé sur la boucle et perçu par tous les noeuds. Toutefois, seul le

noeud comprenant la bonne adresse enverra son accusé de réception à l'émetteur. La complexité de cette implémentation est aussi linéaire.




```
entity NOEUD  is
        port(  Ctrl_noeud       : in   std_logic;
               Adresse_noeud    : in   std_logic_vector (1 downto 0);
               Data_in_m        : in   std_logic_vector (7 downto 0);
               Data_out_m       : out  std_logic_vector (7 downto 0);
               CLK              : in   std_logic);
end NOEUD;
```

Figure 4-11: Schéma synoptique d'un noeud.

La figure 4-11 décrit les ports formant chaque noeud. La description de leurs rôles consécutifs est comme suit:

- Ctrl_noeud: ce port formé d'un seul bit sert à activer ou désactiver le commutateur du noeud. La valeur '1' engendre la communication du signal de l'entrée Data_in_m vers la sortie Data_out_m après un délai inhérent à la technologie utilisée.

- Adresse_noeud: formé par un vecteur de deux bits. Ce champs lit le vecteur path_reseau relatif au medium, et emmagasine les différentes valeurs lues selon leur ordre. Dans la programmation VHDL, les assignations suivantes ont été faites:
    1. Noeud A: 00
    2. Noeud B: 01
    3. Noeud C: 10
    4. Noeud D: 11
- Data_in_m: buffer d'entrée de chaque noeud.
- Data_out_m: buffer de sortie de chaque noeud.
- CLK: port d'entrée du signal d'horloge.

4.4.5 Environnement et description SDS

SDS (System Design Series) est un outil développé par la compagnie Mentor Graphics qui fournit un environnement hautement automatique pour la conception de systèmes électroniques (niveau de conception de systèmes de haut niveau et niveau de conceptions des circuits ASIC). SDS se base sur l'habilité du langage VHDL comme langage de spécification et langage d'implémentation pour lequel, il est capable de représenter un comportement détaillé d'un circuit ASIC numérique.

L'activité de la conception haut niveau du SDS est supportée par un nombre de méthodes abstraites qui définissent l'architecture et le comportement d'un système sans considérer les détails d'implémentation.

Le flux de design de SDS commence par identifier les spécifications du système, développer un modèle de l'environnement du système et un modèle

graphique du système, et conclure par la génération et la simulation du code VHDL (figure 4-13). Une possibilité de générer le code C est offerte également.
On peut décrire la conception avec SDS sur deux niveaux :

1. **Conception au niveau système** : dans ce cas, SDS permet une conception au niveau système avec la possibilité de saisir, d'analyser et de définir les spécifications. Le but est de développer, à partir de ces spécifications, un modèle structurel du système; et de construire, à partir de ce modèle un prototype conceptuel du système qui peut être utilisé afin de vérifier si les spécifications sont bien rencontrées.

2. **Conception ASIC** : dans ce cas, SDS donne au concepteur ASIC l'habilité de produire un modèle comportemental VHDL ou une conception RTL ASIC dans un contexte automatisé. Le modèle VHDL peut être simulé, utilisé pour explorer des architectures alternatives, et utilisé comme une entrée directe pour les outils de synthèse.

La figure (4-13) montre la méthodologie adoptée. En premier lieu, il faut créer un context-diagram de l'architecture désirée. Cette étape consiste à spécifier les différentes entrées et sorties de l'architecture. Ensuite, il s'agit de créer un DFD (Data Flow Diagram) qui décrit l'interaction entre les différents processus du système.

Dans chaque entité du DFD, on spécifie une architecture VHDL qui décrit le comportement du processus. Il serait toujours possible de remplacer cette description VHDL par un FSM. le résultat en sera le même. Le choix entre ces deux possibilités est basé sur divers paramètres, tels la complexité de la conception ou le nombre d'états qui le caractérisent.

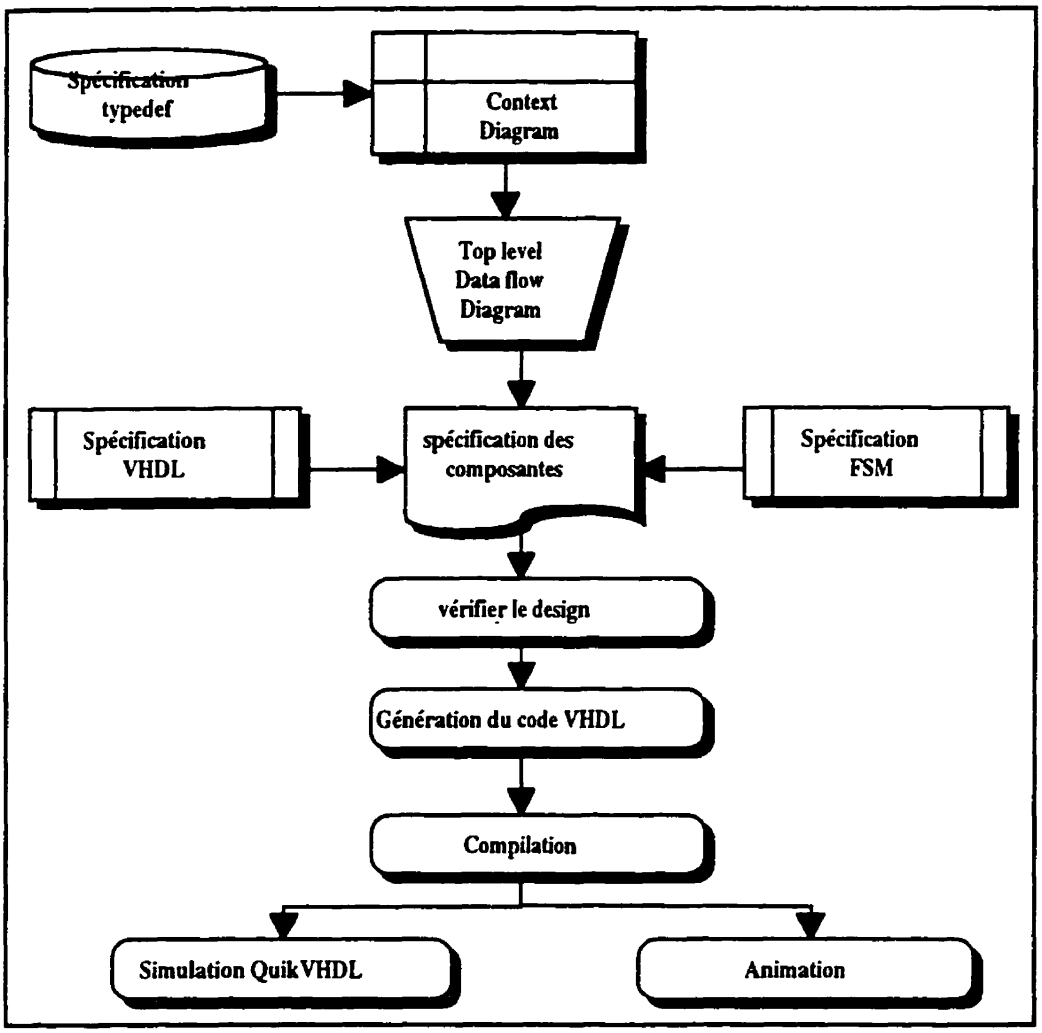


Figure 4-12: Méthodologie de conception avec SDS.

SDS génère une spécification VHDL pour l'ensemble de la conception, tout en incluant les spécifications relatives aux entités déjà décrites. Le code VHDL généré utilise une hiérarchie en assignant a chaque entité dans le DFD une composante qui sera appelée par la suite dans chaque compilation.

Méthodologie suivie pour la description SDS.

L'approche suivie pour la description SDS a été la suivante:

- Générer un context-diagram montrant les signaux d'entrée et de sortie pour tout le système (fig. 4-14).

- Descendre un niveau plus bas pour montrer le diagramme de flux de données DFD illustrant les grands blocs de notre système, à savoir l'émetteur, le medium et le récepteur. D'autres instances doivent être ajoutés sous forme de foreign instance. Leur rôle consiste à commuter les signaux d'entrée/sortie des blocs déjà cités selon les différents besoins. Généralement, leur description se fait sous forme de machine à état FSM.

- Exprimer le medium sous forme de deux parties séparées, une partie matérielle et l'autre logicielle. Pour la partie logicielle, le programme préalablement écrit en C a été 'greffé', tandis que la partie matérielle a été exprimée sous forme de blocs contrôlés par des machines à état.

- Générer un programme VHDL pour les différents blocs déjà énumérés, tout en tenant compte de leurs niveaux de hiérarchie.

- Accomplir les étapes conventionnelles de simulation et de synthèse.

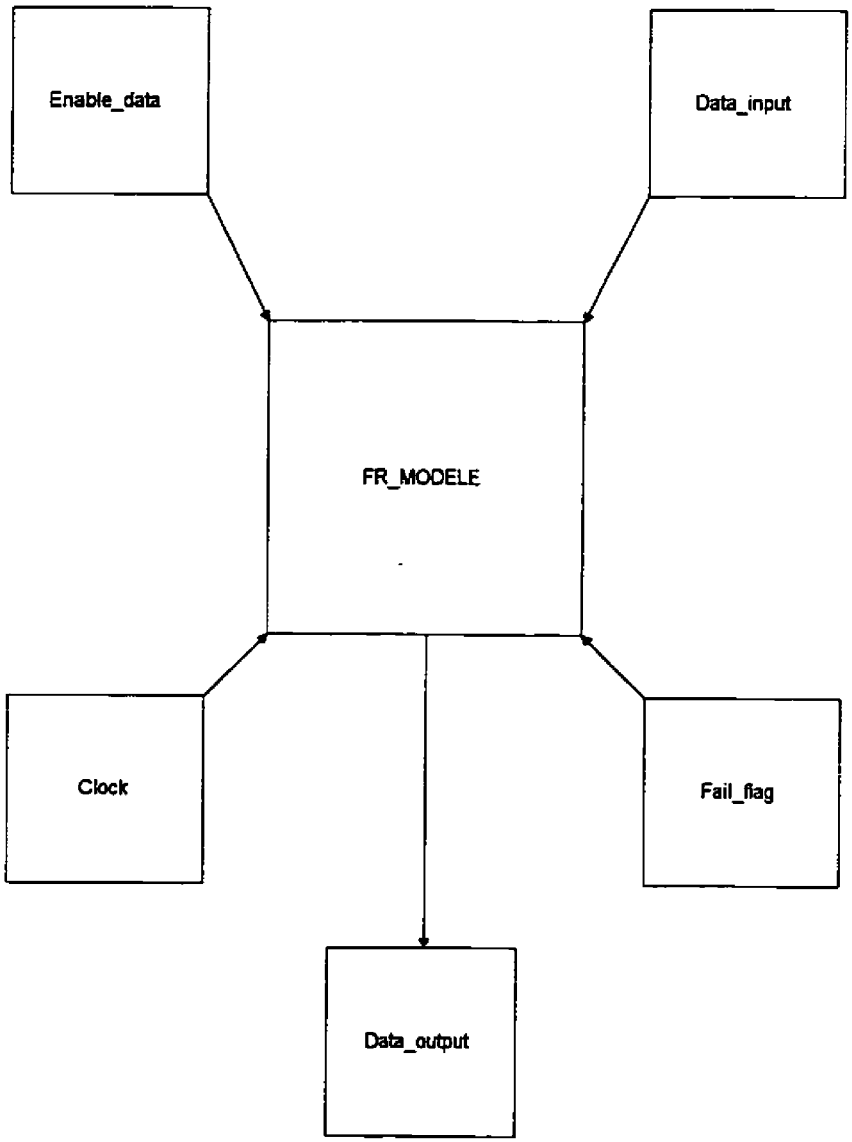


Figure 4-13: Diagramme de contexte de FR_MODELE.

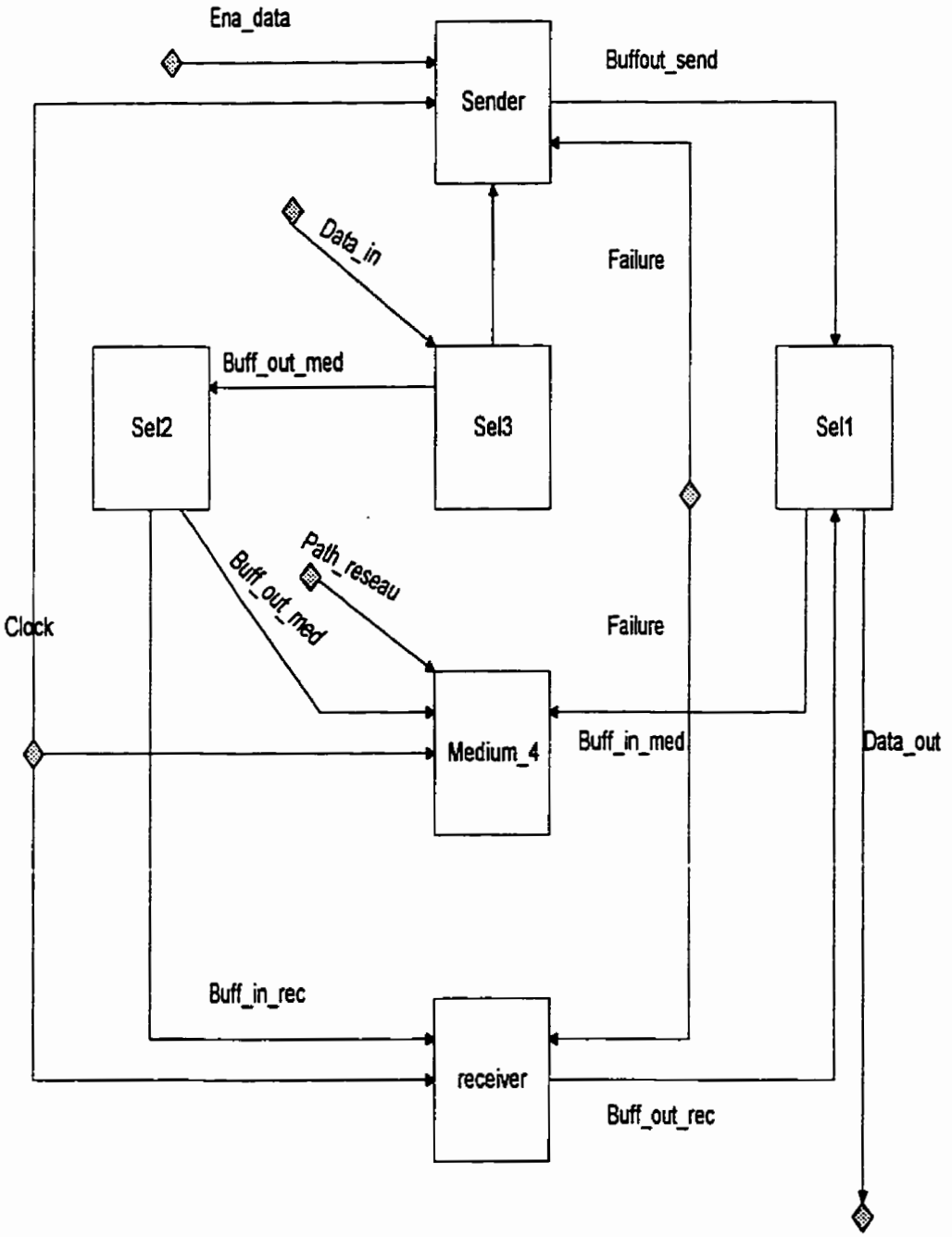


Figure 4-14: Diagramme de flux de données pour le design.

Le chapitre suivant présente les résultats pratiques obtenus avec les descriptions et suppositions faites au chapitre 4. Nous montrerons les résultats des différentes simulations et synthèses ainsi que les performances obtenues.

CHAPITRE 5

RÉSULTATS PRATIQUES

Ce chapitre montre les différents résultats pratiques fournis par ce travail. Il montre la simulation des circuits matériels exécutée par l'outil QuickSim de MentorGraphics. La simulation est effectuée pour chaque circuit individuellement en premier lieu. La simulation de l'ensemble vient couronner le tout.

L'analyse des performances arrive à la suite. Elle vient s'incruster entre le modèle de Frame Relay déjà existant sur le marché et le modèle utopique qui présentera la borne maximale des performances.

La dernière partie sera dédiée à la synthèse des différents circuits sous différentes technologies. Soient le CMOS 1.2 et le FPGA. Une comparaison des différents choix suivra.

5.1 Simulation des circuits matériels

Avant de commencer la simulation, un petit aperçu sur l'analyse des performances et sur les objectifs à atteindre sera donné.

Le circuit doit fonctionner avec une fréquence d'horloge de 100 Mhz. Cette fréquence représente la moyenne des commutateurs existants sur le marché. Cette valeur de fréquence peut être augmentée durant la simulation, un exemple roulant sous 300 Mhz sera ensuite représenté. Toutefois, certains détails de design doivent alors être revus pour garantir une plus grande optimisation du circuit. La description actuelle en mode

comportemental ne permet pas de pousser l'optimisation à des valeurs comparables à celle du mode structurel.
La simulation a pris chaque bloc séparément des autres en premier lieu. L'analyse des performances consistait à ajouter les différents délais de chaque étape.

5.1.1 Simulation de l'émetteur

La simulation de l'émetteur a considéré que ce bloc doit transmettre les signaux du buffer_in_send vers le buffer_out_send en présence de certains signaux de signalisation Frame Relay. Vu la simplification de ce bloc (supression de toutes les options), le temps pris est un cycle d'horloge pour passer de l'entrée à la sortie, d'où T_S = 1 T (figure 5-1), où T représente la période d'horloge.

La simulation montre le fonctionnement de base de l'émetteur à la réception des signaux de signalisation. En effet, elle montre que l'émetteur commence par signaler sa disposition à communiquer avec le medium quand les signaux (Ena_Data = '1' et Clock = '1'). Cet état se traduit par l'émission du signal Setup sur le réseau. Quand l'émetteur reçoit la réponse du medium (Connect) signifiant son envie d'entrer en contact avec lui, il lui répond par le signal (Connect_ack) un coup d'horloge plus tard.

Le signal Failure quant à lui indique la présence d'une anomalie sur le réseau ou d'une défaillance majeure, la simulation montre que sur réception de ce signal et du signal de demande de rupture de liaison (Disconnect) de la part du medium, l'émetteur répond par une confirmation de cette déconnexion (Release), et il retourne alors à son état initial de repos (idle).

Le fichier qui se charge de gérer tous ces signaux est le fichier Assembly.vhd (voir annexe). C'est à lui que revient la tâche de synchroniser le fonctionnement de l'ensemble

des sous fichiers. En fait, ce fichier englobe toutes les fonctionnalités des couches 2 et 3 qui ont été matérialisées.

5.1.2 Simulation du récepteur

Les mêmes remarques faites pour l'emetteur s'appliquent au récepteur. Les signaux du Buff_in_rec sont véhiculés au buff_out_rec après un seul coup d'horloge, d'où T_R = 1 T (figure 5-2).

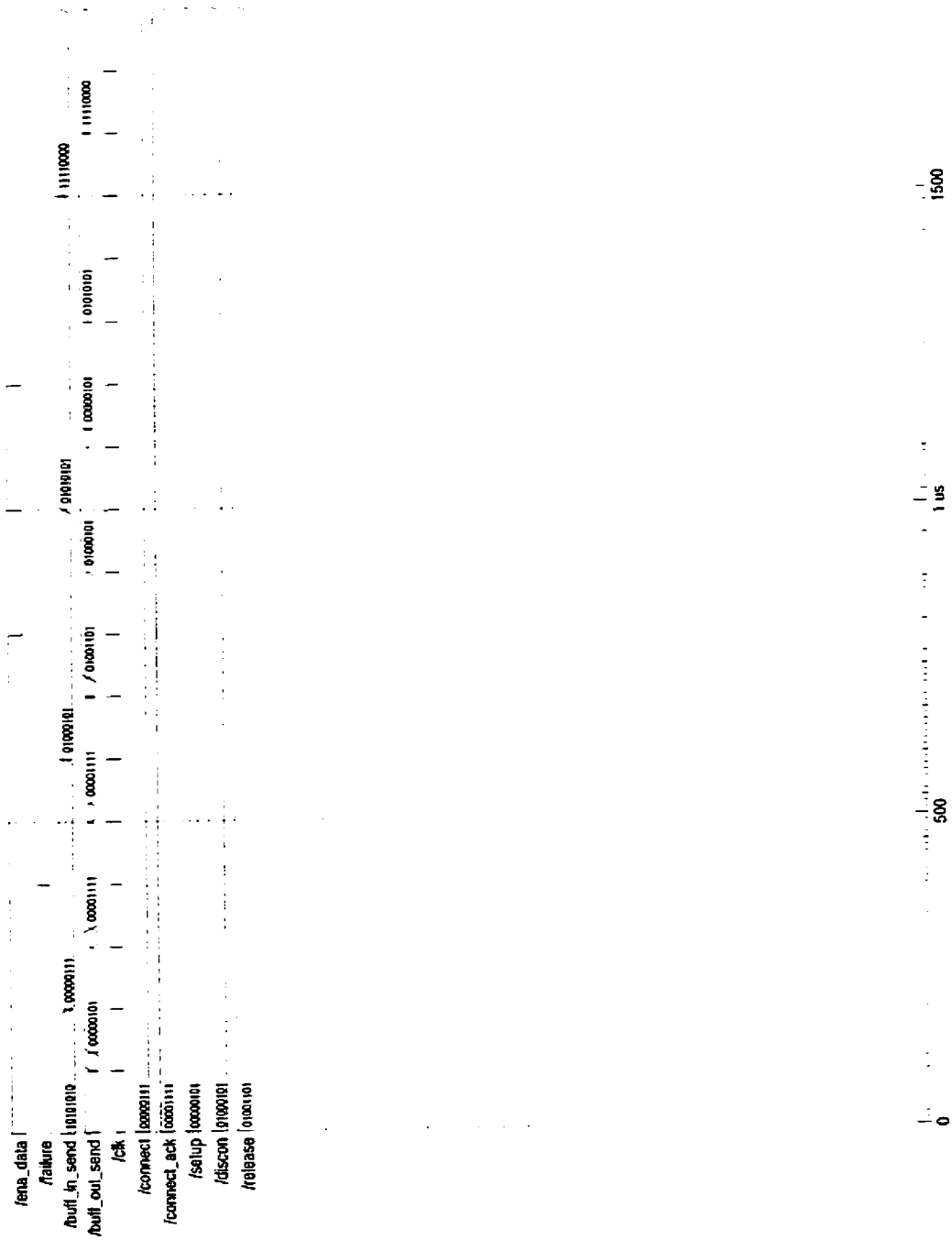


Figure 5-1: Simulation de l'émetteur

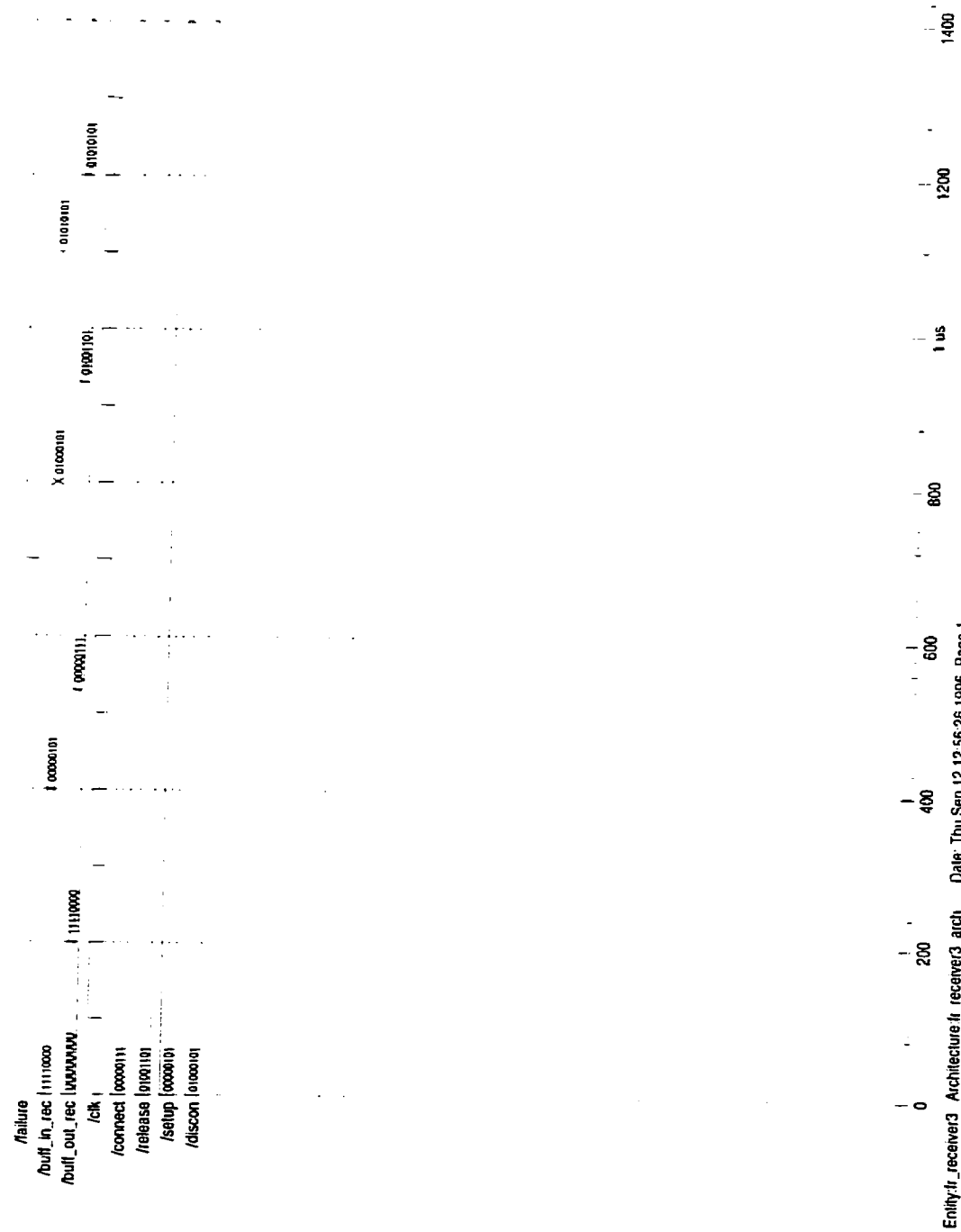


Figure 5-2: Simulation du récepteur

La figure (5.2) montre le fonctionnement de base du récepteur à la réception des signaux de signalisation. En effet, elle montre que le récepteur est initialement en état d'attente de connexion. Il manifeste cette disposition en envoyant le signal *set_up* sur le réseau. Sur réception du signal de demande de connexion (*Connect*) de la part du medium, il répond alors par le signal (*Connect_ack*), fermant ainsi la chaîne de connexion entre les trois blocs: émetteur, medium et récepteur. En conséquence, il devient prêt à recevoir les différentes trames de données envoyées par l'émetteur. Cet état demeurera applicable jusqu'à la réception du signal de défaillance (*Failure)* annonçant une anomalie sur le réseau. Dans ce cas, le récepteur doit attendre la confirmation de la défaillance de la part du medium qui enverra alors le signal (*Disconnect*). Le récepteur répond alors par le signal de confirmation de rupture de liaison (*Release*), et retourne à son état de repos (*idle*).

5.1.3 Simulation de medium

La simulation du medium commence par celle du noeud qui constitue le principal composant. Les résultats de la simulation montrent qu'un cycle d'horloge est nécessaire pour passer les signaux de l'entrée d'un noeud vers sa sortie. $T_N = 1$ T.

Sachant que pour les deux topologies existantes en pratique (bus et boucle) le nombre de noeuds à consulter est de 1 pour le premier cas et n pour le deuxième, nous avons choisi le pire cas où la topologie est celle d'une boucle. Par ailleurs, le nombre de noeuds considéré était 4. Ce nombre est tout à fait arbitraire, et tout autre entier naturel aurait pu le remplacer. Le calcul du temps pris par tout le medium montre que:

$$T_M = (4 \times T_N) + 4\ T \qquad (1)$$

Le deuxième terme de l'addition est relatif au temps de traitement que prend le *router* avant de connecter la sortie d'un noeud vers l'entrée du noeud suivant. Ce qui donne comme temps global pour le medium

$T_M = 8T$ (figure 5-3) (2)

En conséquence, la durée totale de transmission (matérielle) d'un octet sera:

$T_T = T + T + 8T = 10\ T$ (3)

Figure 5-3: Simulation du medium

La simulation de ce bloc a été accomplie en faisant varier le vecteur d'entrée path_réseau. Dans le premier cas, le vecteur a pour valeur '00011110', ce qui correspond au séquencement des noeuds ABDC. Le signal de donnée est d'abord reçu sur le buffer d'entrée du medium (Buff_in_med) et en même temps sur le buffer d'entrée du noeud A qui constitut la source du graphe. Après un coup d'horloge, il est acheminé à la sortie de A. L'adress_buffer_2 lit les deux bits suivants du path_réseau et achemine la donnée vers l'adresse lue, soit le noeud B dans ce cas précis. Un coup d'horloge est également nécessaire pour cette opération.

Le buffer adress_buffer_3 suit le même raisonnement pour véhiculer l'information vers le troisième noeud qui est le D. Le buffer adress_buffer_4 fera de même pour le dernier noeud, le C.

Après un coup d'horloge, les données quittent le dernier noeud pour attaquer le buffer (temp), et un dernier coup d'horloge fera basculer ce signal vers le buffer_out du medium.

5.2 Analyse des performances obtenues

L'analyse de performances a pris comme référence la transmission d'un document de 50 Kb de taille approximative. Le document est formé par un fichier texte de dimensions tout à fait moyennes. Ce texte est converti en un ensemble de trames de 128 bytes chacune, ce qui donne 50 trames pour un texte de 53.2 Kb exactement.

La figure 4.2 montre le chemin parcouru par une trame pour se rendre de l'émetteur vers le récepteur. La figure suivante illustre ce chemin d'une manière plus compréhensible:

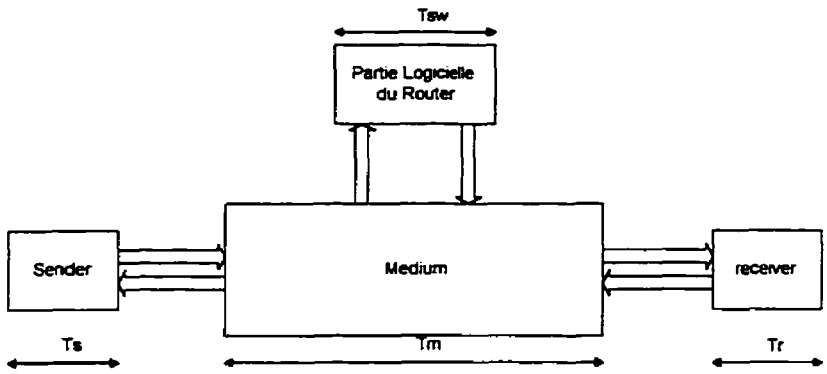


Figure 5-4: Chemin parcouru par une trame.

Rappelons que chaque communication (envoi d'un certain nombre de trames) est toujours précédée de 3 signaux de signalisation Frame Relay (set_up, Connect et Connect_ack). La fin de chaque trame sera suivie de 3 signaux de signalisation (Disconnect, Release et Release_complete).

La figure (4-7) illustre la composition simplifiée d'une trame. Pour notre cas, elle sera formée de 128 octets d'information, de 3 octets d'entête et de 2 octets de queue, ce qui fait un total de 133 octets.

L'analyse de performance prendra trois modèles:

- Modèle Frame Relay conventionnel où le logiciel est consulté par chaque trame.
- Modèle adopté pour ce projet: le logiciel n'est consulté que par la première trame.
- Modèle utopique constituant une borne maximale pour les performances, le logiciel n'est jamais consulté.

5.2.1 Performances du modèle conventionnel Frame Relay

L'équation (2) montre que le temps pris par une trame pour parcourir tout le medium est $T_T = 8T$. Notre analyse de performances se limitera à ce module. Nous ne considérerons pas les temps pris par les modules de l'émetteur et du récepteur.

Temps total pour une trame de 133 octets: 133*8T = 1064 T.

Temps pris par la signalisation: 6 * 8T = 48 T.

Temps mesuré du logiciel: T_{sw} = 30 ms sur une station SPARC -4 .

Temps total T_D pour le document de 50 trames:

$$T_D = [50 * (1064\ T + T_{sw})] + 48T \quad (4)$$

5.2.2 Performances du modèle adopté pour ce projet

Le logiciel n'est consulté que par la première trame, en conséquence

$$T_D = 50\ (1064\ T) + T_{sw} + 48T \quad (5)$$

5.2.3 Performances du modèle idéal

Le logiciel n'est jamais consulté, ce qui donne:

$$T_D = [50 * (1064\ T)] + 48T \quad (6)$$

5.2.4 Exemples numériques

Le tableau (5-1) montre les débits calculés pour différentes valeurs de signaux d'horloge. Pour le premier cas, nous avons pris comme temps de calcul logiciel 30 ms, alors que ce temps a été réduit à 20 ms pour le deuxième cas. Rappelons que la première valeur (30 ms) est une valeur mesurée, alors que la seconde suppose l'utilisation d'un processeur plus performant en vue de déterminer l'impact de l'accélération du temps de traitement logiciel sur le calcul du débit.

	Frame Relay conventionnel		**Modèle de ce projet**			**Modèle idéal**	
T_{SW} = 30 ms							
T (ns)	T_D (ms)	**Débit (Kbps) D_{FR}**	T_D (ms)	**Débit (Mbps) D_{PR}**	**Gain de Débit D_{PR}/D_{FR}**	T_D (ms)	**Débit (Mbps)**
500	1526.6	33.536	56.6	0.904	**26.95**	26.6	1.924
100	1505.32	34.008	35.32	1.449	**42.60**	5.32	9.615
50	1502.66	34.072	32.66	1.567	**45.99**	2.66	19.23
30	1501.6	34.096	31.59	1.620	**47.51**	1.59	32.051
10	1500.5	34.12	30.53	1.677	**49.15**	0.532	96.15
T_{SW} = 20 ms							
500	1.0266	49.872	46.62	1.098	**22.016**	26.6	1.924
100	1.005	50.928	25.32	2.021	**39.68**	5.32	9.615
50	1.0026	51.064	22.66	2.259	**44.23**	2.66	19.23
30	1.00159	51.118	21.597	2.370	**46.36**	1.59	32.051
10	1.000	51.172	20.532	2.493	**48.71**	0.532	96.15

Tableau 5-1: Mesure de débit pour différentes périodes d'horloge

En faisant varier la fréquence d'horloge de 10 ns à 100 ns, les courbes des figures (5.2, 5.3, 5.4 et 5.5) montrent les variations de débits pour chaque modèle. Il en sort que les performances du modèle de ce projet dépendent directement de la fréquence d'horloge utilisée.

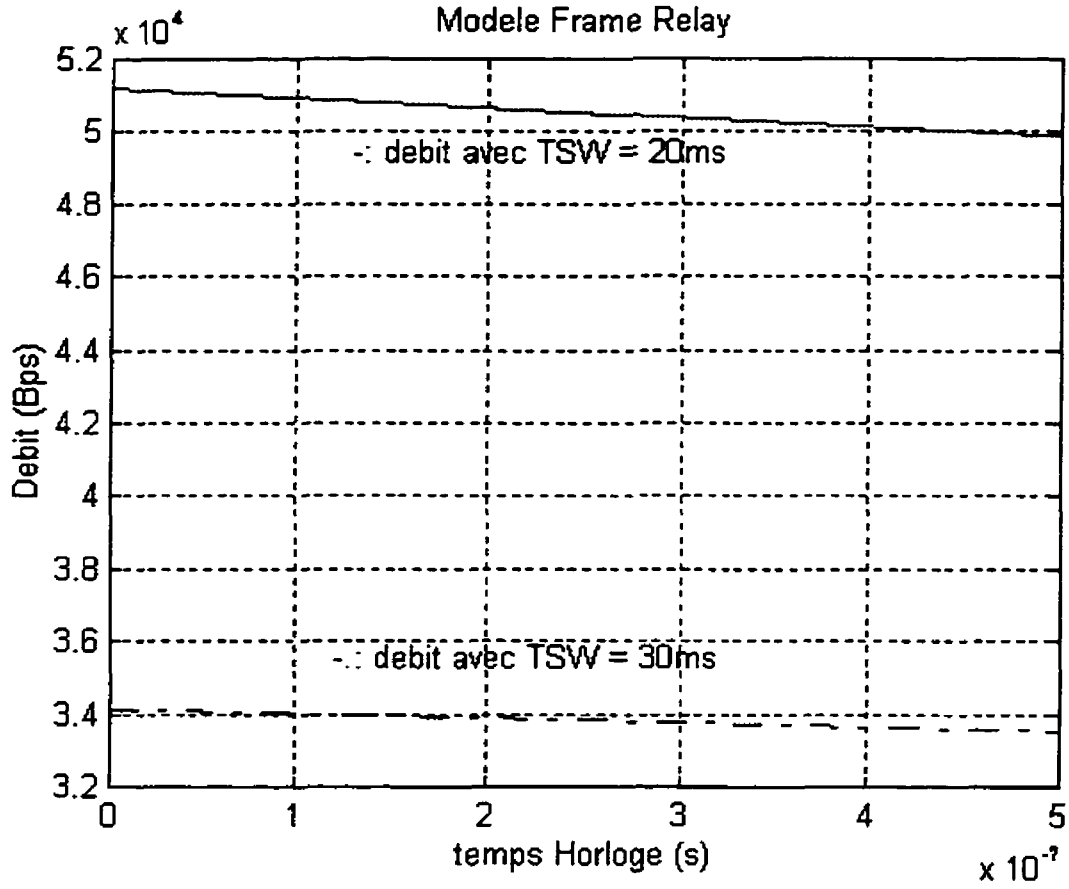


Figure 5-5: Variations de débit pour le frame relay conventionnel

Le modèle conventionnel de Frame Relay est très peu sensible aux variations de la fréquence d'horloge, le temps de déroulement du logiciel occupe environ 280 fois le temps du traitement matériel (pour T = 100 ns). Il serait totalement injustifié d'essayer de maximiser la fréquence d'horloge pour ce type d'architecture.

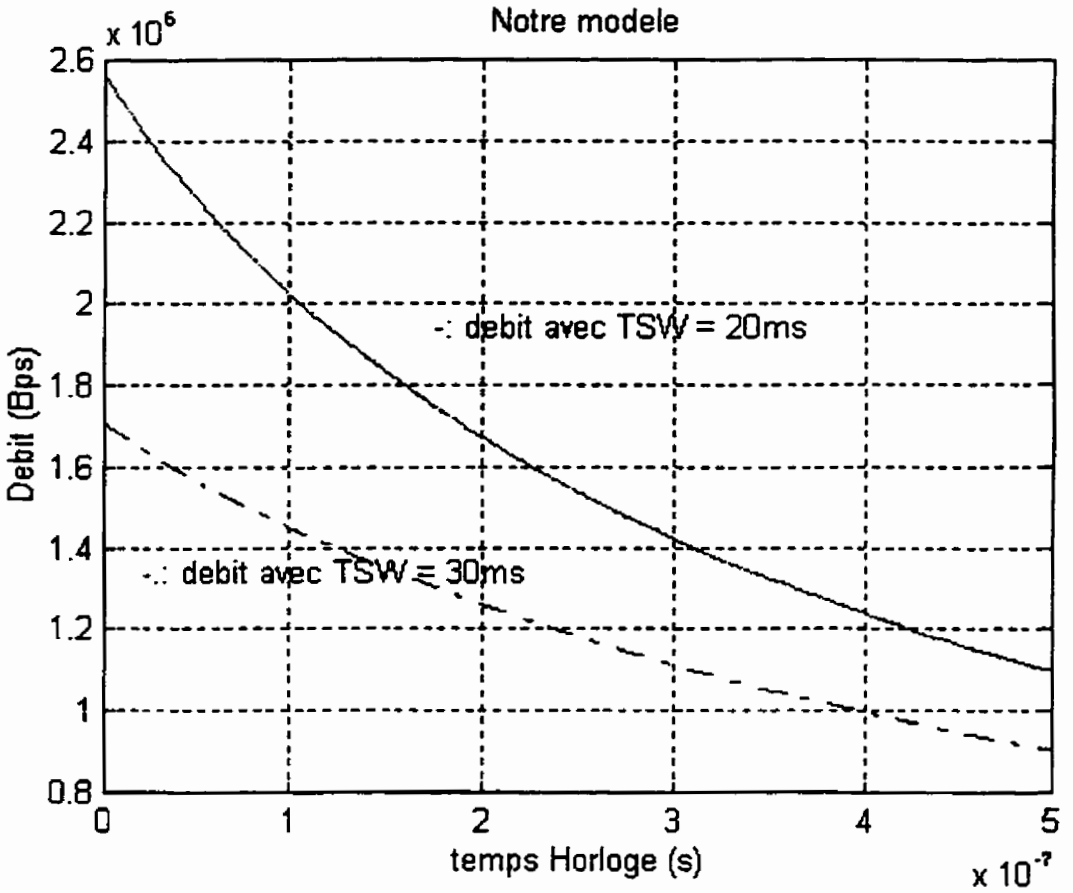


Figure 5-6: Variations de débit pour le modèle de ce projet

Le modèle de ce projet présente une courbe d'évolution de débit qui augmente linéairement avec la fréquence d'horloge. Cet aspect justifie la recherche d'une maximisation du temps d'horloge tout en essayant de concevoir des nouvelles architectures dépendant de moins en moins du logiciel.

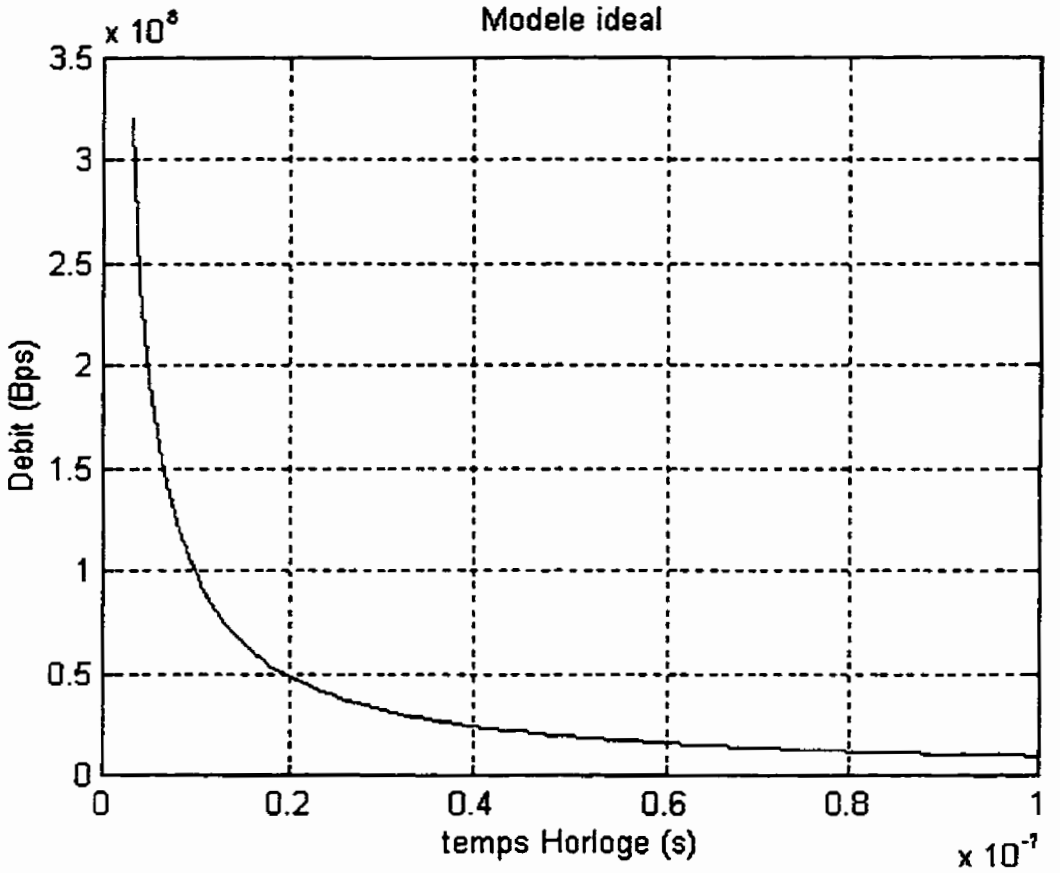


Figure 5-7: Variations de débit pour le modèle idéal

Le modèle idéal ne constitue évidemment qu'une borne maximale pour le calcul de débit, et ne présente qu'une valeur purement indicative. Ceci n'empêche que l'évolution rapide de son débit revête un aspect fort intéressant pour les nouvelles conceptions dans le domaine de co-design.

La courbe suivante montre la variation du gain en débit en fonction du temps d'exécution logiciel considéré.

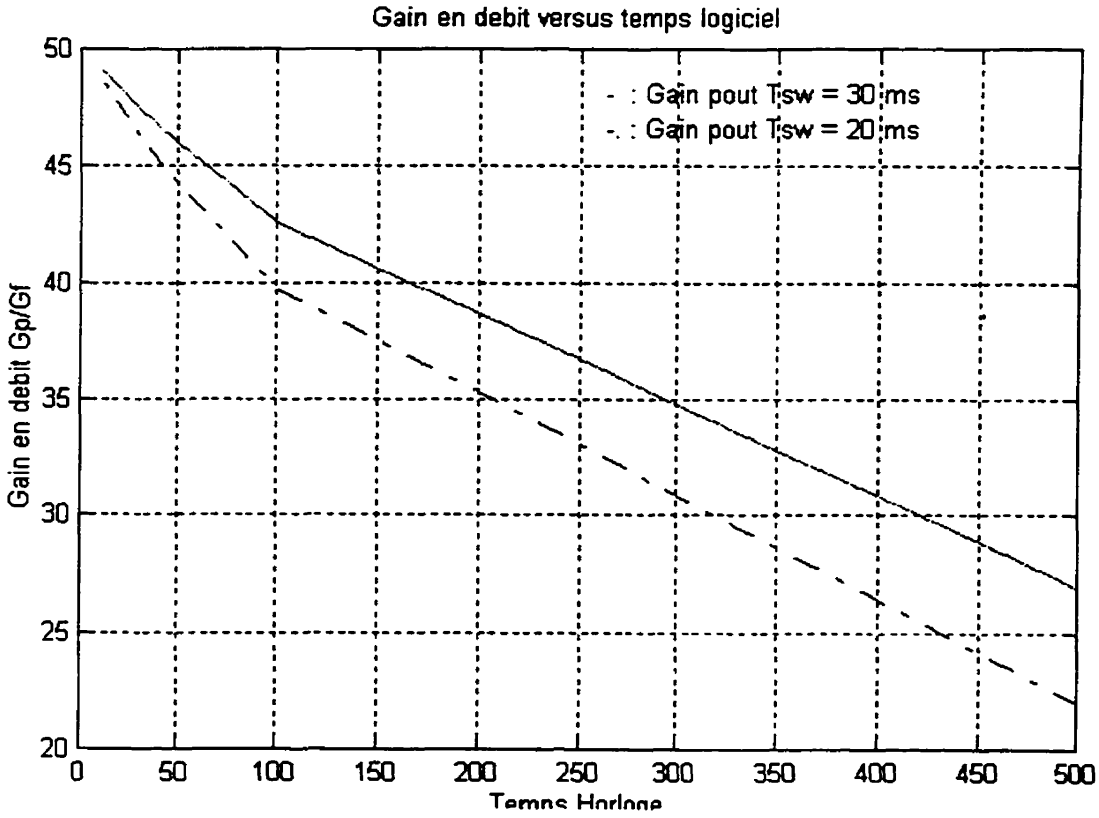


Figure 5-8: Gain en débit versus temps d'exécution du logiciel

Pour les fréquences d'horloge basses, la diminution du temps d'exécution du logiciel procure un gain de débit assez observable. Ce gain de débit diminue proportionnellement à la fréquence de l'horloge utilisée.

L'analyse des courbes précédentes révèle les remarques suivantes:

- Le temps d'exécution du logiciel est un facteur qui influe directement sur le débit des communications réalisé. Sa conséquence diffère selon le modèle adopté. En effet, on remarque que pour le modèle conventionnel de Frame Relay, l'écart de performances reste à peu près constant pour les deux simulations effectuées à des temps d'exécution logiciel différents. En contre partie, pour le modèle de notre projet, cet écart augmente d'une manière inversement proportionnelle au temps d'exécution logiciel, ce qui constitue un facteur incitant à l'accélération du logiciel.
- Le gain en débit est plus marqué pour les grandes valeurs de T quand T_{SW} est grand. Cet écart diminue proportionnellement à T. Pour les petites valeurs de T, nous remarquons que les deux gains convergent vers des valeurs pratiquement identiques.

5.3 Synthèse des circuits générés par VHDL

La synthèse des différents circuits a été accomplis dans divers environnements, à savoir l'Autologic II de MentorGraphics et le Synopsys. Avec le premier outil, nous nous sommes limités à la technologie 1.2 µm CMOS, tandis qu'avec le deuxième, nous avons voulu implémenter en FPGA le résultat de notre synthèse.

5.3.1 Synthèse avec Autologic II

L'Autologic II constitue un environnement de synthèse assez intéressant. Il contient un grand nombre de librairies permettant l'implémentation sous diverses technologies. Dans notre travail, nous avons opté pour le 1.2 um *worst*. Ce choix était

dicté par le souci de vérifier la faisabilité du design et non celui de chercher quelle est la meilleure technologie pour notre architecture.

La synthèse a donné les résultats suivants:

	Avant optimisation					Après optimisation				
	Ports	**Nets**	**Primitives**	**Litweight**	**Surface courante**	**Ports**	**Nets**	**Primitives**	**Litweight**	**Surface courante**
Émetteur	19	53	40	650	312300	19	53	30	190	280300
Récepteur	18	48	36	610	290600	18	36	18	180	224700
Medium	25	810	750	7720	2282200	25	275	142	250	1905300
Noeud	20	46	34	520	268600	20	29	9	200	165900

Tableau 5-2: Résultats de la synthèse avec Autologic II

Optimisation:

L'optimisation doit se passer selon des facteurs pré-définis d'avance, tels la surface, le timing, etc. Notre critère ici était uniquement la surface, car le chronogramme donnait une certaine marge de sécurité. En effet, la fréquence d'opération était fixée à 100 Mhz (T = 10 ns), alors que le chemin critique donnait un délai maximal de 8.9 ns. La fréquence maximale pouvant être atteinte dans nos conditions de travail serait alors:

F = 1/T = 1/(8.9 e-9) => F = 112 359 550 Hz, soit approximativement 110 Mhz.

Il est évident que la recherche d'une plus grande fréquence d'opération doit être basée sur une re-définition des circuits utilisés, une technologie plus performante, etc.

L'Autologic II donne la possibilité de visualiser la sortie schema logique du circuit (Netlist). Pour ce faire, il faut invoquer l'environnement Design_Architect de MentorGraphics. Les schémas figurent à l'annexe.

5.3.2 Synthèse avec Synopsys

La synthèse des circuits visait deux sorties différentes: ASIC et FPGA. Pour le premier cas, l'outil synopsys a fourni des sorties Netlist plus optimisées que son prédécesseur.

L'ASIC procure une surface matérielle plus compacte que le FPGA et une meilleure vitesse de fonctionnement, mais en contre partie il n'est pas re-programmable, ce qui constitue un lourd handicap pour l'utilisation dans un Router.

Les deux choix ont donné des diagrammes de simulation qui les placent dans la limite de performances préalablement fixée, soit une fréquence de 100 Mhz.

5.3.3 Mise en oeuvre des circuits dans un circuit FPGA

Les circuits intégrés programmables FPGA ("Field Programmable Gate Array") combinent les avantages des circuits logiques programmables "Programmable Logic Devices" (PLD) et des matrices de portes "gate arrays". Ils ont les avantages d'avoir jusqu'à 25000 portes logiques tout en étant programmable par l'utilisateur.

L'architecture interne de Xilinx comporte des unités d'entrées-sorties "input-output blocks" (IOB) et des unités logiques configurables "Configurable Logic Blocks" (CLB). Les interconnexions sont réalisées par des matrices de commutations ("Switched Matrix"). Le fonctionnement est basé sur la programmation des cellules internes de mémoires qui vont configurer premièrement la logique et les interconnexions, et deuxièmement les entrées et les sorties. L'organisation générale d'un FPGA est montrée dans la figure (5.9).

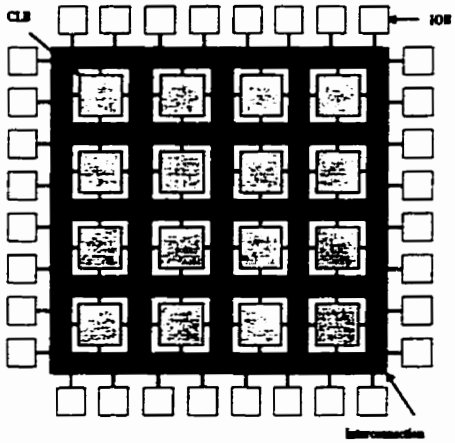


Figure 5-9: Organisation générale d'un FPGA.

Le CLB de Xilinx-XC4000, présenté à la figure 5.7, contient trois générateurs de fonction (F, G, H), deux bascules (SR_X, SR_Y) et plusieurs multiplexeurs programmables.

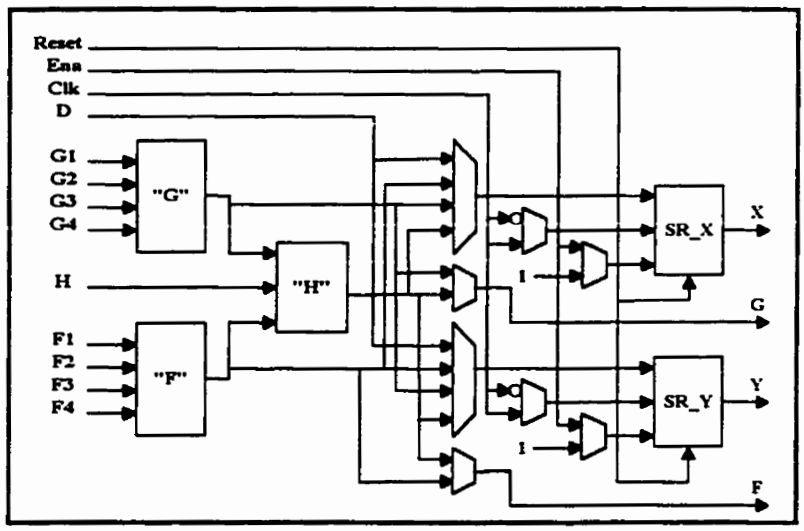


Figure 5-10. Le CLB du Xilinx-XC4000.

<u>Mise en oeuvre des circuits sur Xilinx-XC4010</u>

les 3 circuits émetteur, medium et récepteur ont été compilés avec le compilateur FPGA de Synopsys, la librairie comprend les données du circuit Xilinx-XC4010 (10000 portes logiques au maximum).

Les étapes suivantes sont nécessaires à la mise en oeuvre sur le circuit XC-4010:

1. Synthèse et optimisation du code VHDL avec la bibliothèque Xilinx: cette étape permet la conception des trois circuits avec des portes logiques de la compagnie Xilinx.

2. Placement et routage du circuit: Cette étape permet la programmation des cellules de mémoire dans les CLB et les interconnexions.

Le FPGA a donné un nombre de cellules restreint, permettant d'inclure tout le design matériel en une seule puce. Le tableau suivant (tableau 5-3) illustre les résultats pratiques de notre synthèse.

	ports	Nets	cells	references
Émetteur	19	61	27	14
Récepteur	18	81	41	13
Medium	25	176	103	35
Noeud	20	32	13	6

Tableau 5-3: résultats de la mise en oeuvre sur FPGA.

Le nombre total de cellules est 223. Sachant que le Xilinx-XC4010 contient 400 cellules, le taux d'utilisation sera alors 55 %, ce qui permet bien d'envisager l'implémentation de toute la partie matérielle dans un même circuit FPGA.

5.3.4 Comparaison entre les deux types d'implémentation

Le premier type d'implémentation s'est effectué sur un ASIC en utilisant la technologie CMOS 1.2 tandis que le deuxième s'est effectué sur un FPGA. Chaque solution présente son pour et son contre.

- La première solution permet d'avoir des vitesses de communication plus élevés, et conviendrait certainement à des circuits cherchants des débits très supérieurs à la moyenne de ceux fournis par les LANs conventionnels.
- La deuxième solution quant à elle permet de reconfigurer le réseau selon le nombre d'ajout ou de retrait de stations. Elle permet toute la souplesse requise pour la manipulation des données réelles d'un LAN. Son débit est supérieur à ceux fournis par les implémentations actuelles de Frame Relay, mais il est inférieur à celui d'un ASIC. Mentionnons que son coût de réalisation très inférieur (du moins pour des petites séries) est un atout de taille.

CONCLUSION

Dans ce travail, nous avons proposé un partitionnement pour la réalisation d'un modèle de réseau local opérant sous le protocole de communication Frame Relay. La concrétisation de cette approche servira à étudier l'impact de la mise en oeuvre de certaines parties des algorithmes de routage servant dans les routers ainsi que celle de certaines fonctions inhérentes aux couches 2 et 3 des protocoles de communication.

Notre étude a comporté cinq grandes étapes:

- Revue des concepts de base dans les systèmes de communication.
- Revue des concepts de base pour les procédés de co-design.
- Prépartitionnement des fonctions des trois couches utilisées par Frame Relay.
- Implémentation mixte des parties matérielles et logicielles utilisées.
- Analyse des performances.

Le choix de l'outil de partitionnement ainsi que le langage de description est un paramètre de taille. Après avoir signalé qu'aucun outil de partitionnement ne permet, à l'heure actuelle, d'envisager un usage universel pour le co-design, nous avons opté pour une méthode de prépartitionnement basé sur l'expérience du concepteur. Le choix initial est progressivement raffiné selon les coûts obtenus.

Le langage de description quant à lui constitue un compromis en soi. Le language VHDL ne permet pas d'aller vers des niveaux d'abstraction très élevés, et sa capacité d'exprimer des protocoles de communication devient de plus en plus limitée avec l'accroissement des performances exigées. Le SDS s'est montré très souple pour cet usage, mais son VHDL généré n'est pas optimisé, ce qui exige de la part du concepteur

d'ajouter des sections de code au fur et à mesure de la description schématique de son système.

La conception de la partie logicielle et de la partie matérielle impose des contraintes d'interfacage dont il faut tenir compte. Les deux blocs sont en effet appelés à travailler conjointement et d'une manière synchronisée.

Par ailleurs, la manière à laquelle était conçue notre medium le rend apte à recevoir n'importe quelle topologie de LAN. En effet, la configuration de réseau sous forme de matrice (M x N) permet d'exprimer toutes les topologies réellement existantes. L'adaptation du modèle utilisé dans le cadre précis de ce projet aux topologies les plus courantes (bus ou boucle) se fait en définissant les paramètres du medium lors de la programmation de la partie matérielle.

La principale contribution de ce travail consiste dans la matérialisation partielle des fonctions des couches 2 et 3 d'un Router Frame Relay et dans l'utilisation d'une conception de système permettant la réutilisation de modèles existants de communication. Ceci est obtenu en séparant la communication et le traitement (*computing*) durant toutes les étapes de la conception. Les résultats obtenus reflètent une accélération du débit fort intéressante. Cette accélération est proportionnelle à la fréquence d'horloge utilisée et inversement proportionnelle au temps d'exécution du logiciel, ce qui ouvre le champ aux nouvelles technologies d'implémentations matérielles pour exploiter cet aspect.

RÉFÉRENCES

AIRIAU, R., BERGÉ, J.M, OLIVE, V. et ROUILLARD, J. (1990). VHDL, du langage à la modélisation. Édition ENST.

ANTONIAZZI, S. et MASTRETTI, M. (1990). An Interactive Environment For Hardware-Software System design at the Spécification Level. Microprocessing & Microprogramming, 30, 545-554.

APPLE COMPUTER INC. (1985). Inside AppleTalk.

ASSI, A. (1994). Protocoles de Communication: Modélisation en VHDL pour la Synthèse de Haut Niveau. Mémoire de maîtrise, École Polytechnique de Montréal. Canada.

BARROS, E. et ROSENTIEL, W. (1992). A Method for Hardware/Software Partitioning in Proc of Compeuro. IEEE CS Press, 194-199.

BENNER, T., ERNST, R. et HENKEL, J. (1993). Hardware-Software Cosynthesis for microcontrollers. IEEE Design&Test, 10.

BERTEKAS, D. et GALLAGER, R. (1987). Data Networks. Prentice Hall Inc., New Jersey.

BORIELLO, G., BUCHENRIEDER, K., CAMPOSANO, R., LEE, E., WAXMAN, R. et WOLF, W. (1993). Hardware/Software co-design . IEEE design and test of computers, 83-91.

BUCHENRIEDER, K., SEDLMEIER, A. et VEITH, C. (1993). HW/SW co-design with PRAMs Using CODES. Proc. CHDL, Ottawa, Canada.

BUX, W. (1981). Local Area Subnetworks: A Performance Comparaison. IEEE Trans. On Comm, 10, 1465-1473.

CHIODO, M., GIUSTO, P., JURECSKA, A., LAVAGNO, L., HSIEH, H. et VINCENTELLI, A. (1993). Synthesis of Mixed Software-Hardware Implémentations from CSFM Spécifications. Proc. Of 2nd Workshop on HW/SW co-design, Cambridge, Massachusetts.

DE MICHELI, G. (1990). The Olympus Synthesis System. IEEE design and Test of Computers, Vol.7, 37-53.

DE MICHELI, G. (1994). Synthesis and Optimization of Digital Circuits. McGraw Hill, 54-57.

GAJSKI, D., VAHID, F. et NARAYAN, S. (1994). A Design Methodology For System Spécification Refinement. Proc. Of Edacs, Paris, France.

GIRARD, A. (1996). Routing and Dimensonning in Circuit Switched Networks. Addison Wesley Editors. 34-67.

GUPTA, R. et DE MICHELI, G. (1993). Hardware/Software Cosynthesis for Digital Systems. IEEE design and test of computers, 29-41.

ISMAIL, T., O'BRIEN, K. et JERRAYA, A. (1994). Interactive System-Level Partitioning with partif. Proc. Of Edacs, Paris, France, february.

JERRAYA, A. et O'BRIEN, K. (1994). An Intermediate Format For System-Level Modeling and Synthesis. Computer Aided Software/Hardware Engineering. J.Rozenblit, K.Buchenrieder (eds), IEEE.

KALAVADE, A. et LEE, E. (1993). A Hardware/Software co-design methodology for DSP Applications. IEEE design and test of computers,16-28.

LAGNESE, E.D. et THOMAS, D.E. (1989). Architectural Partitioning for system level Design. 26th ACM/IEEE design Automation Conference, 62-67.

MOUFTAH, H.T. et TAVARES, S.E. (1988). Information Networking: A ten Forecast. Research report submitted to Bell Canada under contract # ORS-88-001, Kingston, Ont., Canada.

SCHWARTZ, M. (1987). Telecommunication Networks. Addison-Wesley Publishing Company.

STALLING, W. (1989). When One LAN Is Not Enough. Byte, vol. 14, 1, 293-298.

Stalling, W. (1996). ISDN and broadband ISDN with Frame Relay and ATM. Prentice Hall, 122-290.

STEINHAUSEN, U. et CAMPOSANO, R. (1993). System Synthesis using Hardware/Software Co-design. Proc. Of 2nd Workshop on HW/SW co-design, Cambridge, Massachusetts.

THOMAS, D.E., ADAMS, J.K. et SCHMITT, H. (1993). A Model and Methodology for Hardware-Software Co-design. IEEE Design and test of Computers, 6-15.

Annexe A: Programme en C pour l'algorithme de Dijkstra

```
/*************************************************************************
*  Sujet:        programme de reduction de coût des matrices              *
*  Fait par:     Haddad Mohamed Tahar                                     *
*  Date :        Juin 1996                                                *
*  Professeurs:  Mr Guy Bois & Mme B. Kaminska                            *
*************************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#define infinie 6300;
typedef struct list
{
 int index, distance;
 struct list *next;          // stucture qui contient un pointeur
}liste, *liste_ptr;    // liste et *liste_ptr correspondent à la même struct

typedef struct node
{
 int index, visiter;
 int distance_min, node_distance_min;
 int arc_entrant,arc_sortant;
 liste_ptr liste_nodes;              //liste_nodes: pointeur du type  liste_ptr
}noeud, *noeud_ptr;                  // noeud_ptr pointe vers node

noeud_ptr *NODE;                     // NODE : pointe vers noeud_ptr qui pointe vers node
void construire_le_graph();
void insert_inlist();
void print_graph();
int chercher_distance_minimum();
int parcourir_le_graph();
void printf_le_chemin_minmum();
int distance_node[11][11];
int nbr_de_nodes;
void  print_noeuds();
main()
{
 int d,i,j,k,l,x,y,cost;             //déclaration des fonctions d'affichage
 char C;
 randomize();
  textbackground(GREEN);
        clrscr();
        textcolor(MAGENTA);
        gotoxy(35,1);
```

```
            cprintf("ROUTING ALGORITHM");
            gotoxy(3,25);
            cprintf(" APPUYER SUR N'IMPORTE QUEL BOUTON POUR QUITTER");

            window(1,2,40,11);
            textbackground(7);
            textcolor(YELLOW);
            clrscr();
            cprintf(" Matrice des couts\n\n\r");

            window(42,2,80,11);
            textbackground(7);
            textcolor(YELLOW);
            clrscr();
            cprintf(" Matrice des couts minima\n\n\r");
            window(1,15,40,24);
            textbackground(7);
            textcolor(YELLOW);
            clrscr();
            cprintf(" Matrice d'adjacence du graphe:\n\n\r");
            window(42,15,80,24);
            textbackground(7);
            textcolor(YELLOW);
            clrscr();
            gotoxy(52,15);
            cprintf("possible  path(s)");
            window(1,1,80,25);

            textbackground(GREEN);
            gotoxy(25,13);
            cprintf("Nombre de noeuds  (de 2 à 8):");
            scanf("%d",&nbr_de_nodes);

/* taille de NODE, tableau de pointeurs = au nombre de neuds */
/*****************************************************************************
*Fonction : tableau pointeur                                                 *
*Traitement: tableau pointeur = nombre de noeuds                             *
*****************************************************************************/

NODE = (noeud_ptr *) malloc(sizeof(noeud_ptr)*nbr_de_nodes);
x=5;
y=4;
C ='A';

for(i=0; i < nbr_de_nodes; i++)
{
   gotoxy(x,y);
   cprintf("%c",C);
    x+=4;
    j =  (int) C;
    C = (char)(j + 1);
```

```
    }
    x=3;
    y=5;
    C ='A';
    for(i=0; i < nbr_de_nodes; i++)
  {
     gotoxy(x,y);
     cprintf("%c",C);
      y+=1;
      j =  (int) C;
      C = (char)(j + 1);
  }

   x=5;
   y=5;
   /*** Lecture des couts **/
/*****************************************************************************
*Fonction: distance_node                                                      *
*But: lecture des couts de depart                                            *
*Traitement: lire les donnees telles qu'affichees dans la matrice de depart  *
*****************************************************************************/
 for(i=0; i < nbr_de_nodes; i++)
 {
  for(j=0; j < nbr_de_nodes; j++)
   {
    gotoxy(x,y);
    cost = random (10);                          /*le coût doit absolument être positif*/
    distance_node[i][j] = cost;
    cprintf("%d ",cost);
     x+=4;
    }
    y+=1;
    x = 5;
 }
 /*****************************************************************************
 * Initialization et allocation de l'espae memoire des noeuds                  *
 *****************************************************************************/

 for(i=0; i <nbr_de_nodes; i++)
  {
   NODE[i] = (noeud_ptr) malloc(sizeof(noeud));
   NODE[i]->arc_entrant = 0;
   NODE[i]->arc_sortant = 0;
   NODE[i]->index = i;
   NODE[i]->liste_nodes = NULL;
  }
/*****************************************************************************
*Fonction: chemins de longueurs                                               *
*But affiche l'evetuelle existence d'une liaison entre deux points donnees    *
*****************************************************************************/
 construire_le_graph();
```

```
print_noeuds(4,16);
x=6;
y=17;
 for(i=0; i < nbr_de_nodes; i++)
{
 for(j=0; j < nbr_de_nodes; j++)
 {
   gotoxy(x,y);
   if(i==j)
        cprintf("0");
        else if(distance_node[i][j] > 0)
         cprintf("1");
        else
          cprintf("0");
         x+=4;
 }
 y+=1;
 x =6;
}
/*****************************************************************************
*Fonction: matrice des couts minimums                                        *
*But: afficher le chemin le plus court entre deux points donnes              *
*****************************************************************************/
print_noeuds(44,4);
x=46;
y=5;
for(i=0; i < nbr_de_nodes; i++)
{
 for(j=0; j < nbr_de_nodes; j++)
   {
   gotoxy(x,y);
        if(i==j)
         cprintf("0");
        else
         {
          k = chercher_distance_minimum(i,j);
          cprintf("%d",k);
         }
         x+=4;
   }
  y+=1;
  x = 46;
 }
/*****************************************************************************
*Fonction qui illustre les differents raccourcis pris pour aller de A        *
*vers un autre noeud du graphe                                               *
*****************************************************************************/
 C='A';
 cprintf("\n");
 x=43; y=16;
 for(j=1;j<nbr_de_nodes;j++)
```

```
          {
           i =0;
           d =  (int) C;
           C = (char)(d + 1);
           gotoxy(x,y);
    cprintf("\npassage de A à %c:  ",C);
    y++;
     if(i==j)
          printf(" 0 ");
         else
          {
           k = chercher_distance_minimum(i,j);
           printf_le_chemin_minmum(i);
          }
         }
   scanf("%d",&i);


}
/*****************************************************************************
 printf_le_chemin_minmum
*****************************************************************************/
void printf_le_chemin_minmum(k)
    int k;
{
 NODE[k];
 printf(" %d ",k);
 for(;;)
   {
    if( NODE[k]->distance_min == 0)
          break;
     printf("=>%d ",NODE[k]->node_distance_min);

          k = NODE[k]->node_distance_min;
   }
}
/*****************************************************************************
*Fonction: construire le graphe                                              *
*But: inclure tous les noeuds connectes à notre noeud de depart              *
*REMARQUE:                                                                   *
*peut importe ce qu'on a chosi comme distance  au départ entre le noeud a et le *
**noeud a, distance = 0;                                                     *
*Si c'est -1, distance  = infinie.                                           *
*Pour chaque noeud A, il va inclure dans sa liste chainee tous               *
*les autres noeuds qui lui sont connectes                                    *
*****************************************************************************/


void construire_le_graph()
{
 int i,j;                          /*procede recursif*/
```

```
 for(i = 0; i < nbr_de_nodes; i++){
  for(j=0; j < nbr_de_nodes; j++){
   if(j != i && distance_node[i][j] != -1){
        insert_inlist(i,j,distance_node[i][j]);
   }
  }
 }
}

/*****************************************************************************
*Dans cette procedure on incremente les compteurs de l'arc intrant et del'arc sortant *
*Fonction: insert_liste                                                        *
*But: dresse la liste des nodes connectees à un noeud donné                    *
*Traitement: incrementation compteurs intrants et extrants                     *
*Entrees: liste_nodes                                                          *
*Sortie: nouvelle_node                                                         *
*****************************************************************************/


void insert_inlist(i,j,dist)
   int i,j,dist;
{
 liste_ptr nouvelle_node;
 nouvelle_node = (liste_ptr) malloc(sizeof(liste));
 nouvelle_node->index = j;
 nouvelle_node->distance = dist;
 nouvelle_node->next = NODE[i]->liste_nodes;
 NODE[i]->liste_nodes = nouvelle_node;
 NODE[i]->arc_entrant++;
 NODE[j]->arc_sortant++;
}

/*****************************************************************************
*Procedure pour chercher le cout minimum pour deux noeud quelconques           *
*Fonction:distance_min                                                         *
*But: minimiser la distance entre deux points consideres                       *
*Traitement: supposer la distance initiale infinie, proceder ensuite par       *
*            successions de comparaisons pour ne garder finalement que         *
*            la plus petite des distances possibles                            *
*****************************************************************************/
int chercher_distance_minimum(k,l)                    /** distance pour aller de k à l **/
   int k,l;
{
 int i;

 if(k== l)
  return -1;
 /* initialisation du graphe */

 for(i=0; i <  nbr_de_nodes; i++){
  NODE[i]->distance_min = infinie;
```

```
   NODE[i]->visiter = 0;
   }

   NODE[l]->distance_min = 0;
   parcourir_le_graph(k,l);
   }

int parcourir_le_graph(k,l)
    int k,l;
{
 liste_ptr liste_courante;
 int poid;
 if(NODE[k]->visiter == 0)
   {
    NODE[k]->visiter = 1;                    /* pour vister une seule fois un noeud du graphe */

    if(k==l)
         return 0;                                              /***** la distance est 0 **/
    else
         {
         /* prendre d.min*/

          for(liste_courante = NODE[k]->liste_nodes; liste_courante != NULL;
             liste_courante  = liste_courante->next)
           {
             poid = parcourir_le_graph(liste_courante->index,l);
             poid += liste_courante->distance;
             if(poid < NODE[k]->distance_min)
                  {
                   NODE[k]->distance_min = poid;
                   NODE[k]->node_distance_min = liste_courante->index;
                  }
           }
          return NODE[k]->distance_min;
         }
   }
 else
   return NODE[k]->distance_min;  /*prendre la plus petite des distances*/
}
/*****************************************************************************
*Fonction print_noeuds(a,b)
*Cette fonction sert à imprimer les differents noeuds du graphe. Elle est
*souvent invoquee dans le present programme.
*****************************************************************************/
 void  print_noeuds(a,b)
  int a,b;
  {
 int i,j,k,l,x,y;
 char C;
 x=a+2;
 y=b;
```

```
 C ='A';
 for(i=0; i < nbr_de_nodes; i++)
 {
    gotoxy(x,y);
    cprintf("%c",C);
     x+=4;
     j =  (int) C;
     C = (char)(j + 1);
 }
 x=a;
 y=b+1;
 C ='A';
 for(i=0; i < nbr_de_nodes; i++)
 {
    gotoxy(x,y);
    cprintf("%c",C);
     y+=1;
     j =  (int) C;
     C = (char)(j + 1);
 }
}
```

Annexe B: Declaration VHDL d'un noeud

```
library IEEE, ARITHMETIC;
use IEEE.std_logic_1164.all;
use ARITHMETIC.std_logic_arith.all;


entity FR_MEDIUM is
        port(  Ctrl_noeud       : in  std_logic;
               Adresse_noeud    : in  std_logic_vector (1 downto 0);
               Data_in_m        : in  std_logic_vector (7 downto 0);
               Data_out_m       : out std_logic_vector (7 downto 0);
               CLK              : in  std_logic);



end FR_MEDIUM;

architecture RESEAU_1f of FR_MEDIUM is


begin
-----

-------- Process du choix du noeud ----------------------
-----------------------------------------------------------
process (CLK)

        begin
      if (CLK'event and (CLK = '1') and (CLK'last_value = '0')) then

              if (Ctrl_noeud = '1') then
                Data_out_m <=Data_in_m after 10 ns;-- sachant que f = 100 Mhz
              end if;
         end if;

        end process;

end RESEAU_1f;
```

Annexe C: Declaration VHDL du sender

```
__*****************************************************************
-- Filename   : frame_sender                                        *
-- Title      : VHDL description of fr_sender entity                *
__*****************************************************************
library IEEE, ARITHMETIC;
use IEEE.std_logic_1164.all;
use ARITHMETIC.std_logic_arith.all;


__ ****************************************************
-- Declaration des ports par lesquels l'entite sender          *
-- communique avec son environnement.                           *
__ ****************************************************


entity FR_SENDER3 is
  port ( Ena_data        : in std_logic:= '0';
         Failure         : in std_logic:= '0';
         BUFF_IN_SEND    : in std_logic_vector (7 downto 0);
         BUFF_OUT_SEND   : out  std_logic_vector (7 downto 0)        ;
         CLK             : in std_logic);
end FR_SENDER3;

architecture FR_SENDER3_ARCH of FR_SENDER3 is
      --signal idle          : std_logic_vector (7 downto 0):= "00000000";

      signal  connect        : std_logic_vector (7 downto 0):= "00000111";
      signal  connect_ack    : std_logic_vector (7 downto 0):= "00001111";
      signal  setup          : std_logic_vector (7 downto 0):= "00000101";
      signal  discon         : std_logic_vector (7 downto 0):= "01000101";
      signal  release        : std_logic_vector (7 downto 0):= "01001101";


begin
SENDER : process(Ena_data, CLK, Failure, BUFF_IN_SEND, connect, connect_ack, setup)
  begin

__ ***********************************************************
-- ici l'entite sender attend une transaction sur un des          *
-- ports qui changent (ou laissent tel quel) son etat. A la       *
-- reception d'une transaction le sender agit en consequence      *
-- et se mette a l'etat indique par cette transaction.            *
__ ***********************************************************
      BUFF_OUT_SEND <= BUFF_IN_SEND after 100 ns;


if (Ena_data = '1') then
```

```
    if ( CLK = '1' and CLK'event) then
        BUFF_OUT_SEND <= setup after 30 ns;
            if (BUFF_IN_SEND = connect) then
                BUFF_OUT_SEND <= connect_ack after 30 ns;
            end if;

-- Disconnect request or failure
            if (Failure = '1') then
              if (BUFF_IN_SEND = discon)then
                BUFF_OUT_SEND <= release after 30 ns;
              end if;
            end if;
  end if;

end if;



      end process ;
end FR_SENDER3_ARCH;
```

Annexe D: Declaration VHDL du receiver

```
_*****************************************************************
-- Filename   : frame_receiver                                     *
-- Title      : VHDL description of fr_receiver entity             *
_*****************************************************************
library IEEE, ARITHMETIC;
use IEEE.std_logic_1164.all;
use ARITHMETIC.std_logic_arith.all;


-- ******************************************************
-- Declaration des ports par lesquels l'entite receiver   *
-- communique avec son environnement.                     *
-- ******************************************************

entity FR_RECEIVER3 is
  port ( Failure          : in std_logic:= '0';
         BUFF_IN_REC      : in  std_logic_vector (7 downto 0);
         BUFF_OUT_REC     : out std_logic_vector (7 downto 0);
         CLK              : in std_logic);
end FR_RECEIVER3;

architecture FR_RECEIVER3_ARCH of FR_RECEIVER3 is


         signal  connect    : std_logic_vector (7 downto 0):= "00000111";
         signal  release    : std_logic_vector (7 downto 0):= "01001101";
         signal  setup      : std_logic_vector (7 downto 0):= "00000101";
         signal  discon     : std_logic_vector (7 downto 0):= "01000101";



 begin
RECEIVER : process(CLK, setup, BUFF_IN_REC, CONNECT, FAILURE, DISCON, RELEASE)
  begin

-- ************************************************************
-- ici l'entite receiver attend une transaction sur un des    *
-- ports qui changent (ou laissent tel quel) son etat. A la   *
-- reception d'une transaction le sender agit en consequence  *
-- et se mette a l'etat indique par cette transaction.        *
-- ************************************************************
if ( CLK = '1' and CLK'event) then
        BUFF_OUT_REC <= BUFF_IN_REC after 100 ns;


  if (BUFF_IN_REC = setup ) then
```

```
                        BUFF_OUT_REC <= connect after 30 ns;
  end if;


-- Disconnect request or failure
        if (Failure = '1'  or discon = "01000101" ) then

                        if (BUFF_IN_REC = discon)then
                                BUFF_OUT_REC <= release after 30 ns;
                        end if;
        end if;

end if;

        end process ;
end FR_RECEIVER3_ARCH;
```

Annexe E: Declaration VHDL du medium

```
library IEEE, ARITHMETIC;
use IEEE.std_logic_1164.all;
use ARITHMETIC.std_logic_arith.all;



-- Graphe comprenent 4 noeuds. A est l'entree (source)
-- il s'agit de lire le vecteur qui est retourne par le software C (8 bits) et qui
-- comprend les adresses des noeuds a connecter(A=00, B=01, C 10, D=11)
-- Le signal data_in doit voyager de A vers les autres noeuds jusqu'a la sortie


entity fr_medium4 is
        port(  path_reseau      : in  std_logic_vector (7 downto 0);
               BUFF_in_MED      : in  std_logic_vector (7 downto 0);
               BUFF_out_MED     : out  std_logic_vector (7 downto 0);
               CLK              : in std_logic);
end fr_medium4 ;

architecture RESEAU4_ARCH of fr_medium4 is

component fr_medium
        port(  Ctrl_noeud       : in  std_logic;
               Adresse_noeud    : in  std_logic_vector (1 downto 0);
               Data_in_m        : in  std_logic_vector (7 downto 0);
               Data_out_m       : out std_logic_vector (7 downto 0);
               CLK              : in  std_logic);


end component;


---------- Configuration des composantes ----------

   FOR NOEUD_A : fr_medium    USE ENTITY work.fr_medium(reseau_1f);
   FOR NOEUD_B : fr_medium    USE ENTITY work.fr_medium(reseau_1f);
   FOR NOEUD_C : fr_medium    USE ENTITY work.fr_medium(reseau_1f);
   FOR NOEUD_D : fr_medium    USE ENTITY work.fr_medium(reseau_1f);


---- signaux intermediaire et alias ----------

               signal ctrl_n_A  : std_logic :='1';
               signal ctrl_n_B  : std_logic :='1';
               signal ctrl_n_C  : std_logic :='1';
               signal ctrl_n_D  : std_logic :='1';
```

```
--
                signal adresse_A        : std_logic_vector (1 downto 0) :="00";
                signal adresse_B        : std_logic_vector (1 downto 0) :="01";
                signal adresse_C        : std_logic_vector (1 downto 0) :="10";
                signal adresse_D        : std_logic_vector (1 downto 0) :="11";
--
                signal Data_out_A       : std_logic_vector (7 downto 0);
                signal Data_out_B       : std_logic_vector (7 downto 0);
                signal Data_out_C       : std_logic_vector (7 downto 0);
                signal Data_out_D       : std_logic_vector (7 downto 0);
--
--
                signal Data_in_A        : std_logic_vector (7 downto 0) :="00000000";
                signal Data_in_B        : std_logic_vector (7 downto 0) :="00000000";
                signal Data_in_C        : std_logic_vector (7 downto 0) :="00000000";
                signal Data_in_D        : std_logic_vector (7 downto 0) :="00000000";

--



--
                signal temp             : std_logic_vector (7 downto 0) ;
                signal adress_buffer_1  : std_logic_vector (1 downto 0) := "00";
                signal adress_buffer_2  : std_logic_vector (1 downto 0) := "00";
                signal adress_buffer_3  : std_logic_vector (1 downto 0) := "00";
                signal adress_buffer_4  : std_logic_vector (1 downto 0) := "00";
--
begin
-----

-------- Process du choix du noeud ----------------------
-----------------------------------------------------------
adresse_A <= "00";
adresse_B <= "01";
adresse_C <= "10";
adresse_D <= "11";
DATA_IN_A     <= BUFF_in_MED;
adress_buffer_4 <= path_reseau(7 downto 6);
adress_buffer_3  <= path_reseau(5 downto 4);
adress_buffer_2 <= path_reseau(3 downto 2);
adress_buffer_1 <= path_reseau(1 downto 0);


noeud_A : FR_MEDIUM
        port map ( ctrl_n_A, adresse_A, DATA_IN_A, data_out_A, CLK);
noeud_B : FR_MEDIUM
        port map ( ctrl_n_B, adresse_B, data_in_B , data_out_B,  CLK);
noeud_C : FR_MEDIUM
        port map ( ctrl_n_C, adresse_C, data_in_C , data_out_C, CLK);
```

```
noeud_D : FR_MEDIUM
        port map ( ctrl_n_D, adresse_D, data_in_D,  data_out_D, CLK);


affect : process (CLK,PATH_RESEAU, ADRESS_BUFFER_1,
ADRESS_BUFFER_2,ADRESS_BUFFER_3, ADRESSE_B, DATA_OUT_A, DATA_OUT_B,
ADRESSE_C, DATA_OUT_C, ADRESSE_D, DATA_OUT_D, TEMP ,BUFF_in_MED)

begin
-----


    if (CLK'event and CLK = '1') then

                if (adress_buffer_3  = adresse_B) then
                            data_in_B <= data_out_A after 10 ns;
                        if (adress_buffer_2  = adresse_C) then
                                    data_in_C <= data_out_B after 10 ns;
                                    if (adress_buffer_1  = adresse_D) then
                                            data_in_D <= data_out_C after 10 ns;
                                            temp      <= data_out_D after 30 ns;
                                    end if;

                        else      data_in_D <= data_out_B after 10 ns;
                                    if (adress_buffer_1  = adresse_C) then
                                            data_in_C <= data_out_D after 10 ns;
                                            temp      <= data_out_C after 30 ns;
                                    end if;
                        end if;

-------------

                elsif (adress_buffer_3 = adresse_C) then
                            data_in_C <= data_out_A after 10 ns;
                        if (adress_buffer_2  = adresse_B) then
                                    data_in_B <= data_out_C after 10 ns;
                                    if (adress_buffer_1  = adresse_D) then
                                            data_in_D <= data_out_B after 10 ns;
                                            temp      <= data_out_D after 30 ns;
                                    end if;

                        else      data_in_D <= data_out_C;
                                    if (adress_buffer_1  = adresse_B) then
                                            data_in_B <= data_out_D after 10 ns;
                                            temp      <= data_out_B after 30 ns;
                                    end if;

                        end if;
---------------

                else  data_in_D <= data_out_A after 10 ns;
                        if (adress_buffer_2  = adresse_B) then
                                    data_in_B <= data_out_D after 10 ns;
```

```
                                        if (adress_buffer_1  = adresse_C) then
                                                data_in_C <= data_out_B after 10 ns;
                                                temp      <= data_out_C after 30 ns;
                                        end if;

                        else    data_in_C <= data_out_D after 10 ns;
                                        if (adress_buffer_1  = adresse_B) then
                                                data_in_B <= data_out_C after 10 ns;
                                                temp      <= data_out_B after 30 ns;
                                        end if;

                        end if;
-----------
                end if;

                BUFF_out_MED <= temp;

        end if;


end process;
-----------


end RESEAU4_ARCH;
```

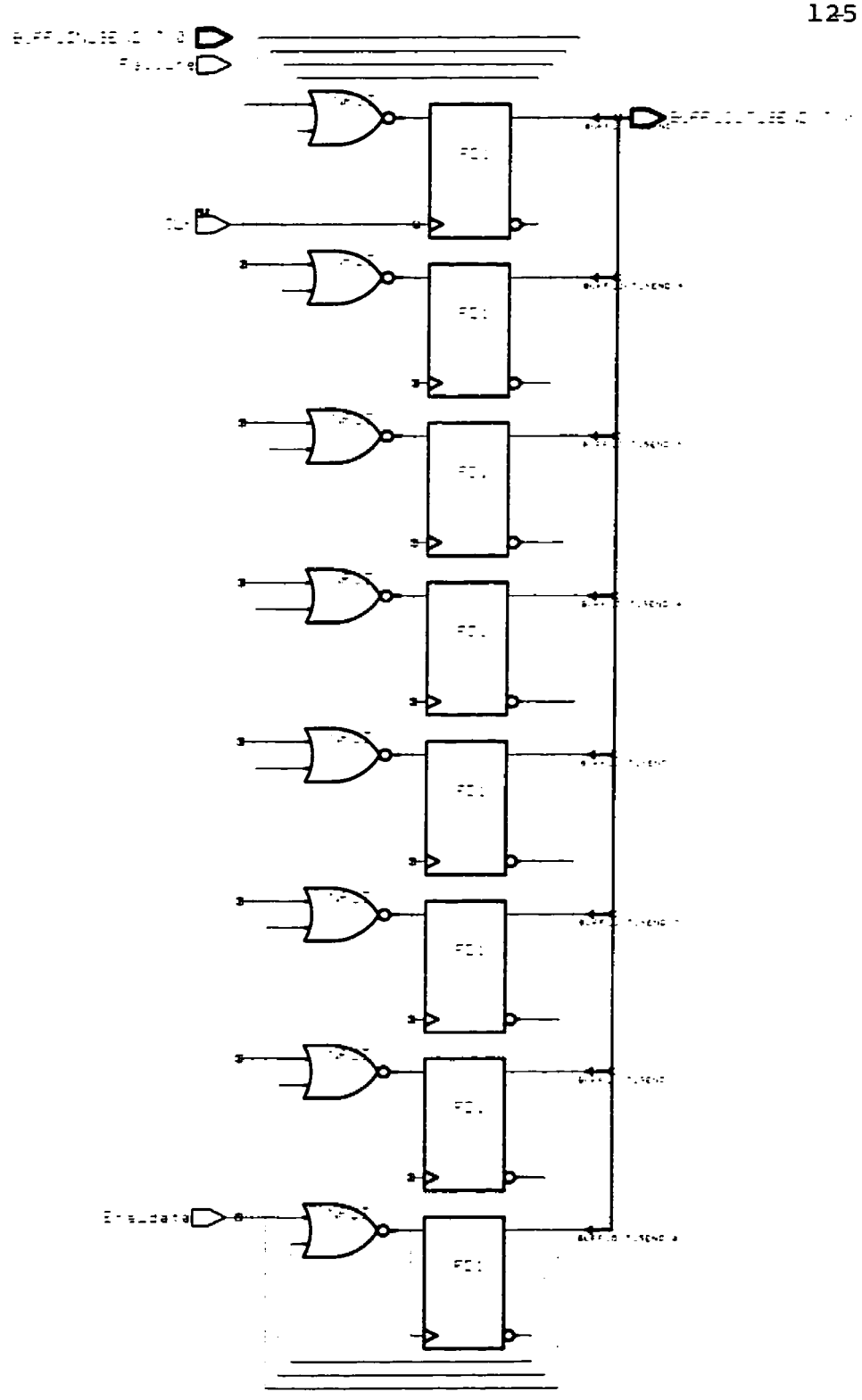
Annexe F: Declaration VHDL du makefile

```
--*****************************************************************
-- Filename   : frame_medium                                      *
-- Title      : VHDL description of fr_medium entity              *
--*****************************************************************
library IEEE, ARITHMETIC;
use IEEE.std_logic_1164.all;
use ARITHMETIC.std_logic_arith.all;


entity FRAME_RELAY is
end FRAME_RELAY;


architecture FR_ARCH of FRAME_RELAY is

component FR_SENDER3
  port ( Ena_data        : in std_logic:= '0'                          ;
         Failure         : in std_logic:= '0'                          ;
         BUFF_IN_SEND    : in std_logic_vector (7 downto 0) ;
         BUFF_OUT_SEND   : out std_logic_vector (7 downto 0)      ;
         CLK             : in std_logic)                               ;

end component;

component FR_MEDIUM4
      port (path_reseau  : in  std_logic_vector (7 downto 0);
         BUFF_in_MED     : in  std_logic_vector (7 downto 0);
         BUFF_out_MED    : out  std_logic_vector (7 downto 0);
         CLK             : in std_logic);
end component;

component FR_RECEIVER3
  port ( Failure         : in std_logic:= '0'                          ;
         BUFF_IN_REC     : in  std_logic_vector (7 downto 0);
         BUFF_OUT_REC    : out  std_logic_vector (7 downto 0)      ;
         CLK             : in std_logic)                               ;
end component;


for all : FR_SENDER3        use entity WORK.FR_SENDER3(FR_SENDER3_ARCH)   ;
for all : FR_MEDIUM4        use entity WORK.FR_MEDIUM4(RESEAU4_ARCH)   ;
for all : FR_RECEIVER3      use entity WORK.FR_RECEIVER3(FR_RECEIVER3_ARCH);

         signal c           :  integer;
         signal CLK         :  std_logic;
         signal Ena_data    :  std_logic := '0';
         signal Failure     :  std_logic := '0';
```

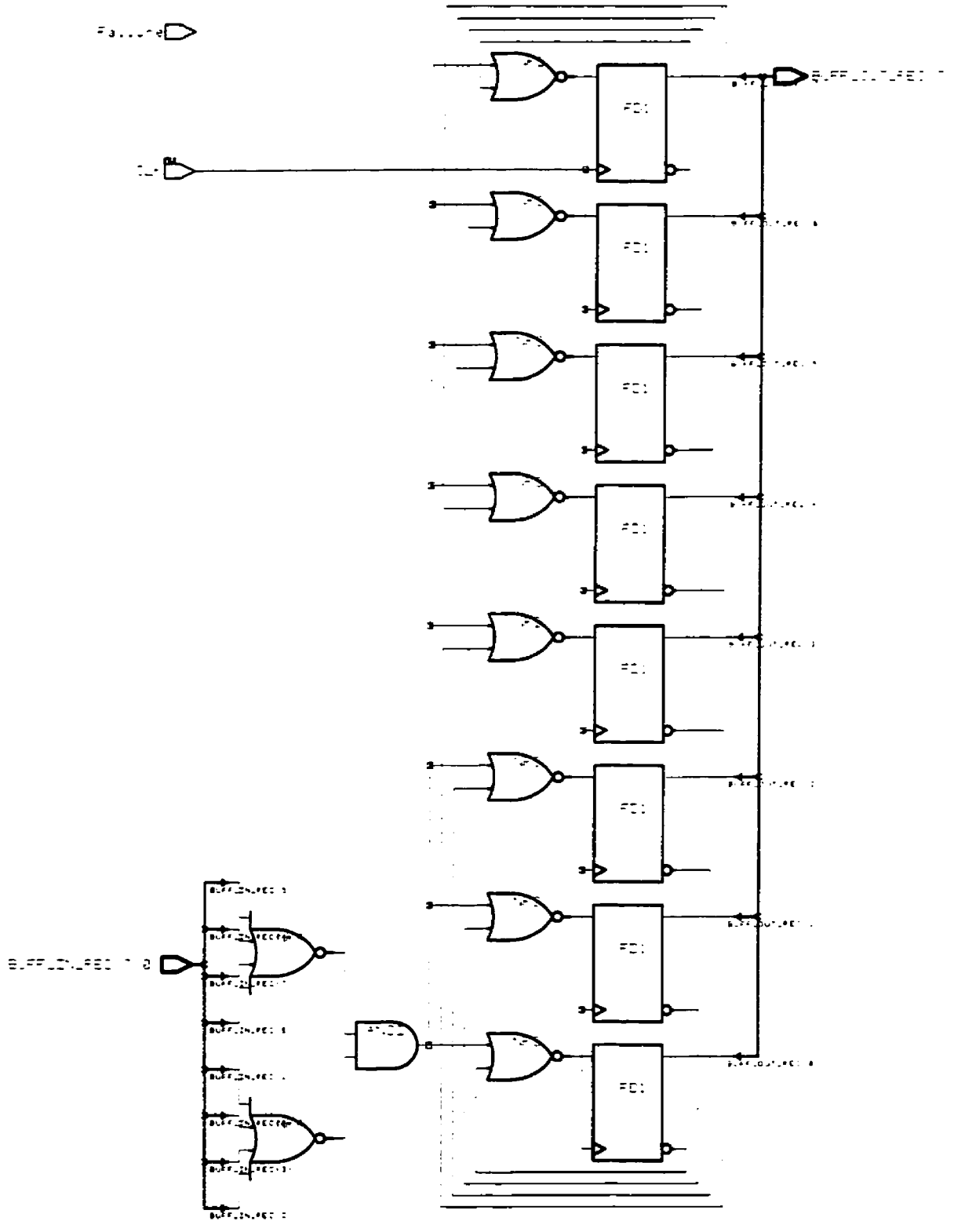
```
signal Data_in              :  std_logic_vector (7 downto 0);
signal Data_out             :  std_logic_vector (7 downto 0);
signal path_reseau          :  std_logic_vector (7 downto 0);
signal BUFF_IN_SEND  :  std_logic_vector (7 downto 0):= "00000000";
signal BUFF_OUT_SEND:  std_logic_vector (7 downto 0):= "00000000";
signal BUFF_IN_REC   :  std_logic_vector (7 downto 0):= "00000000";
signal BUFF_OUT_REC  :  std_logic_vector (7 downto 0):= "00000000";
signal BUFF_in_MED   :  std_logic_vector (7 downto 0):= "00000000";
signal BUFF_out_MED  :  std_logic_vector (7 downto 0):= "00000000";

signal connect              :  std_logic_vector (7 downto 0):= "00000111";
signal release              :  std_logic_vector (7 downto 0):= "01001101";
signal connect_ack          :  std_logic_vector (7 downto 0):= "00001111";
signal setup                :  std_logic_vector (7 downto 0):= "00000101";
signal discon               :  std_logic_vector (7 downto 0):= "01000101";
signal release_complete:  std_logic_vector (7 downto 0):= "01011010";
signal adress               :  std_logic_vector (7 downto 0):= "01100000";
signal FCS                  :  std_logic_vector (7 downto 0):= "00000110";
signal Flag_in              :  std_logic_vector (7 downto 0):= "01111110";
signal Flag_out  :  std_logic_vector (7 downto 0):= "01111110";

begin

        ---------------------------------
        -- Creation des exemplaires des entites --
        ---------------------------------

SENDER3 : FR_SENDER3 port map (      Ena_data, Failure,BUFF_IN_SEND,BUFF_OUT_SEND,
CLK);

MEDIUM4 : FR_MEDIUM4 port map (path_reseau, BUFF_in_MED,BUFF_out_MED, CLK);

RECEIVER3 : FR_RECEIVER3 port map ( Failure, BUFF_IN_REC, BUFF_OUT_REC, CLK);

--*************************************************************************************
commute: process (CLK,Ena_data,PATH_RESEAU, SETUP, CONNECT, CONNECT_ACK, FAILURE,
DISCON, BUFF_in_MED, BUFF_out_MED,RELEASE, RELEASE_COMPLETE)

begin
-------------------------------------------------------------------
--premiere connexion: sender to medium
-------------------------------------------------------------------

        if (CLK = '1' and CLK'event) then
           BUFF_OUT_SEND <= setup;
           BUFF_in_MED  <= setup;
--BUFF_OUT_REC
-------------------------------------------------------------------
-------------------------------------------------------------------
-- deuxieme connexion: Medium to receiver
-------------------------------------------------------------
```

```
                BUFF_out_MED <= setup;
                BUFF_IN_REC  <= setup;
                if (BUFF_in_MED = connect) then
                        if (BUFF_out_MED = connect_ack) then
                                BUFF_out_MED <= CONNECT;
                                BUFF_in_SEND<= CONNECT;
                                if (BUFF_OUT_SEND=CONNECT_ACK) then
                                  BUFF_in_MED<=CONNECT_ACK;
                                end if;
                        end if;

                end if;


--*************************************************************************
-- A ce niveau, on est en droit de supposer que lien physique est etabli
-- entre le sender et le receiver.
--*************************************************************************
 for c in 0 to 132 loop
    if( c= 0 ) then
          BUFF_OUT_SEND<= Flag_in;
    elsif (c = 1) then
          BUFF_OUT_SEND<= adress;
    elsif (c = 2) then
          BUFF_OUT_SEND<= adress;
    elsif (c = 131) then
          BUFF_OUT_SEND<= FCS;
    elsif (c = 132) then
          BUFF_OUT_SEND<= Flag_out;
    end if;
BUFF_IN_SEND <= Data_in;
 BUFF_in_MED <=BUFF_OUT_SEND after 200 ns;
BUFF_IN_REC <= BUFF_out_MED after 200 ns;
DATA_OUT <= BUFF_OUT_REC after 200 ns;
end loop;
-------------------------------------------------------------------
-- coupure de la deuxieme connexion: Medium to receiver
-------------------------------------------------------------------

          if (Failure = '1'  or Ena_data = '0' ) then
                  BUFF_in_MED <= discon;
           if (BUFF_out_MED = release) then
              BUFF_in_MED <= release_complete;
           end if;

          end if;

-------------------------------------------------------------------
-- coupure de la premiere connexion: Sender to Medium
-------------------------------------------------------------------
```

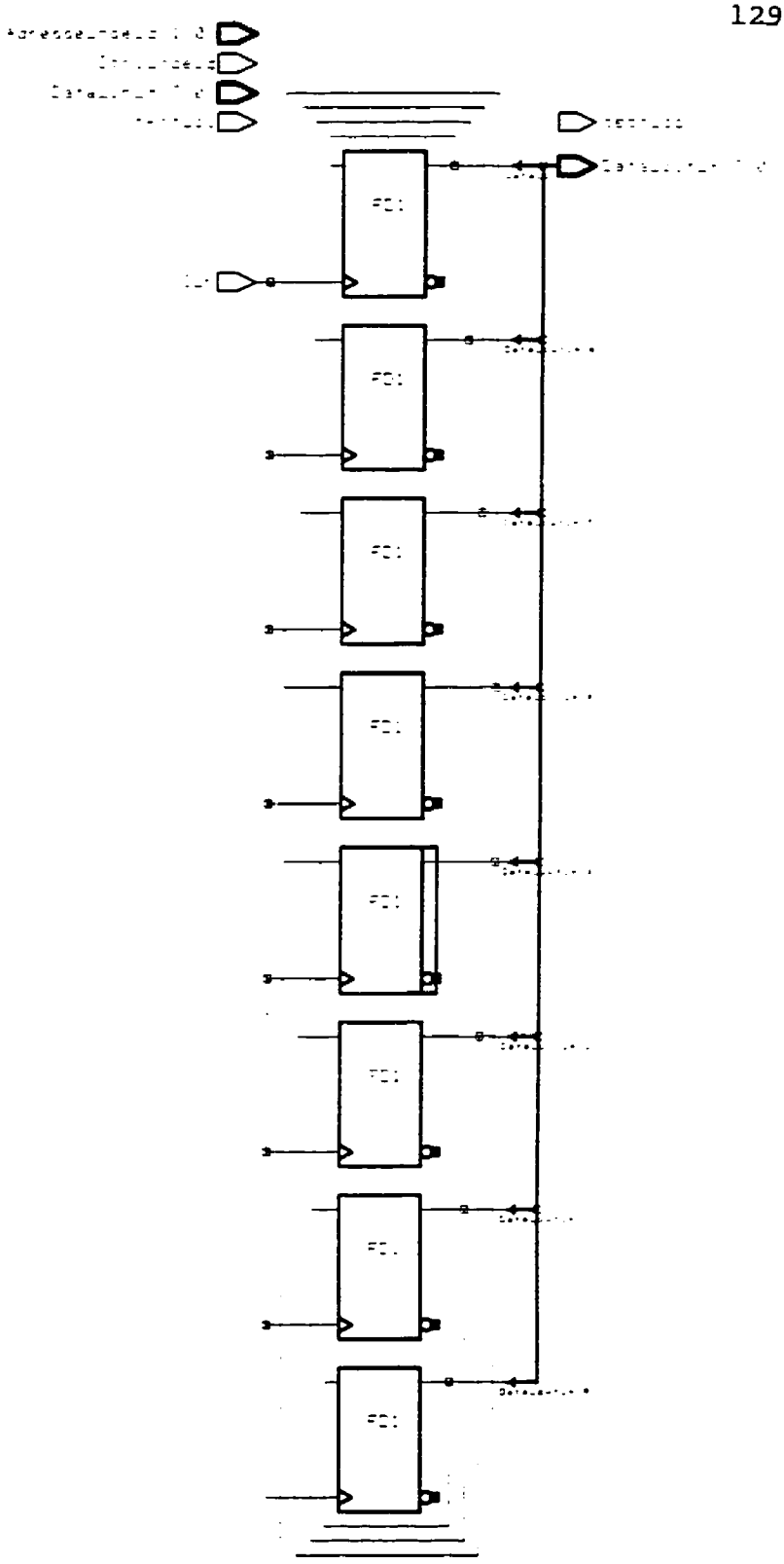
```
        if (Failure = '1'  or  Ena_data = '0' ) then
          if (BUFF_OUT_SEND = discon) then
                BUFF_IN_SEND <= release;
            if (BUFF_OUT_SEND = release_complete) then
                path_reseau <= "00000000";
            end if;
          end if;
        end if;

--*****************************************************************************************
        end if;

end process;

end FR_ARCH;
```



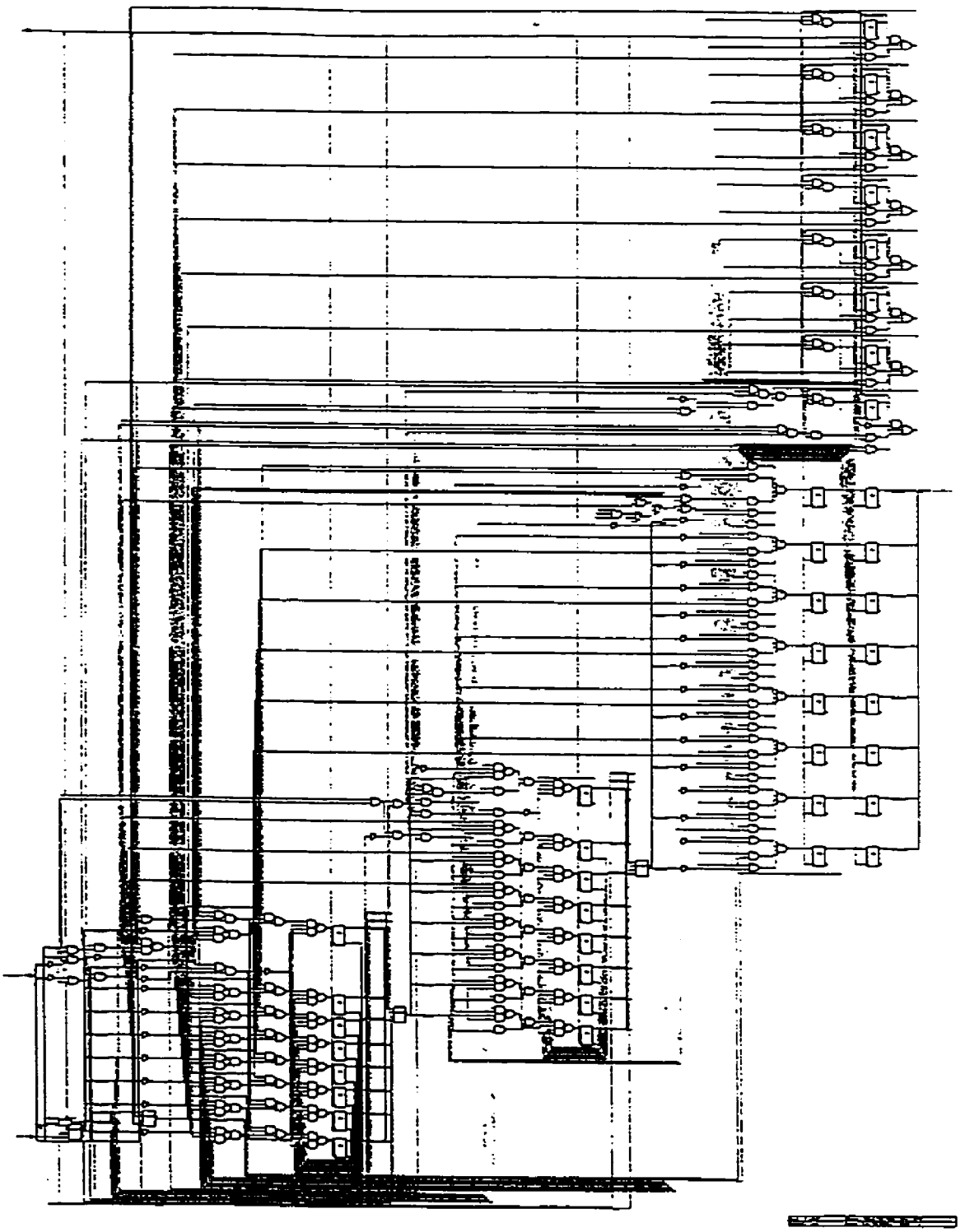
design FF_SEN3	designer Tamar Haddad	date 10 10 96
technology tsmc	company Ecole Polytechnique	sheet 1 of 1



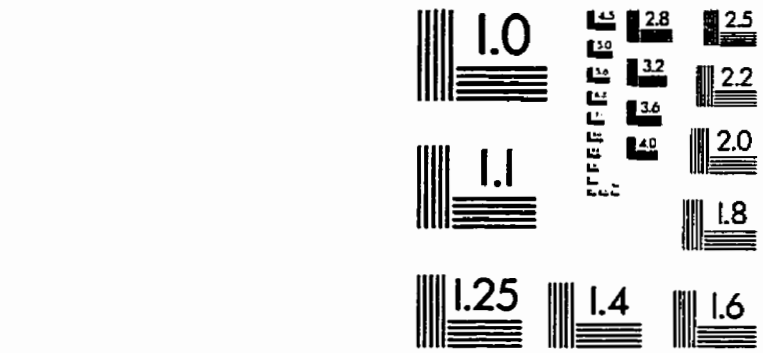
design FF_RECEIVER	designer Tahar Haddad	date 10 12 96
technolog. class	compan. Ecole Polytechnique	sheet 1 of 1

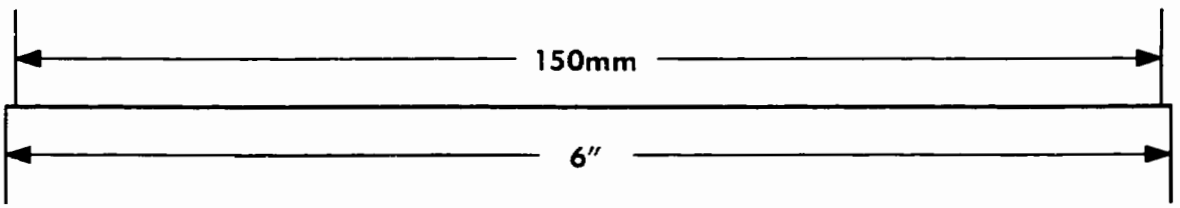


design: FPLMEDIUM_Interface	designer: Michael Tamar	date: 3/3/96
technology: class	company: Ecole Polytechnique	sheet: 1 of 1



TEST TARGET (QA-3)





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989